НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

«До захисту допущено»
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки: 6.040301 «Прикладна математика»
(код і назва)

на тему: Задача відновлення S-блоку за таблицею розподілу диференціалів відносно модульного додавання,

Виконав (-ла): студент (-ка) 4 курсу, групи ФІ-52
(шифр групи)

Єршов Степан Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: доцент кафедри ММЗІ, к.т.н. Яковлєв С. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант: _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент: _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
Фізико-технічний інститут

Кафедра математичних методів захисту інформації

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки - 6.040301 «Прикладна математика»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедрою

М.М.Савчук

_____
(підпис) (ініціали, прізвище)

«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту

Єршов Степан Олександрович
(прізвище, ім'я, по батькові)

1. Тема роботи Задача відновлення S-блоку за таблицею розподілу диференціалів відносно модульного додавання,

керівник роботи доцент кафедри ММЗІ, к.т.н. Яковлєв С.В._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _____ р. № _____

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи узагальнення властивостей таблиць розподілу диференціалів; алгоритм відновлення S-блоку за таблицею розподілу диференціалів відносно модульного додавання,

4. Зміст роботи дослідження аналітичних властивостей таблиць розподілу диференціалів та задача відновлення S-блоку за таблицею розподілу диференціалів відносно модульного додавання; розробка алгоритму відновлення S-блоку за таблицею розподілу диференціалів,

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	провести огляд опублікованих джерел за тематикою дослідження	05.10.18	
2.	сформувати та довести аналітичні властивості таблиць розподілу диференціалів відносно модульного додавання	30.12.18	
3.	сформувати та проаналізувати задачу відновлення S-блоку за таблицею розподілу диференціалів	28.03.19	
4.	запропонувати алгоритм відновлення на основі підходу Backtracking та експериментально його перевірити.	25.05.19	

Студент _____ Єршов С. О.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Яковлєв С. В.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Кваліфікаційна робота містить: 80 стор., 1 рисунок, 5 таблиць, 9 джерел.

Метою є розробка нових підходів та методів до аналізу криптографічних примітивів, що дозволить створювати нові надійні системи криптографічного захисту інформації та оцінювати стійкість існуючих.

Об'єктом дослідження є інформаційні процеси в системах захисту інформації.

Предметом дослідження є алгебраїчні властивості криптографічних нелінійних перетворень.

В цій роботи розглянуто проблему відновлення S-блоків за таблиці розподілу диференціалів (далі DDT) в умовах модульного додавання. Досліджено структуру спеціальних класів еквівалентності для S-блоків малої бітності (переважно чотирьохбітові та менше). Розглянуто класи алгебраїчних перетворень, що не змінюють вид DDT. Розроблено алгоритм відновлення спеціального класу еквівалентності за конкретною DDT.

S-БЛОК, ТАБЛИЦЯ РОЗПОДІЛУ ДИФЕРЕНЦІАЛІВ, КЛАС DDT-ЕКВІВАЛЕНТНОСТІ, ВІДНОВЛЕННЯ S-БЛОКА ЗА DDT, МОДУЛЬНЕ ДОДАВАННЯ

РЕФЕРАТ

Дипломная работа содержит: 80 стр., 1 рисунок, 5 таблиц, 9 источников.

Целью является разработка новых подходов и методов к анализу криптографических примитивов, что позволит создавать новые надежные системы криптографической защиты информации и оценивать устойчивость существующих.

Объектом исследования являются информационные процессы в системах защиты информации.

Предметом исследования является алгебраические свойства криптографических нелинейных преобразований.

В данной работе рассмотрена проблема восстановления S-блоков по таблице распределения дифференциалов (далее DDT) в условиях модульного сложения. Исследована структура специальных классов эквивалентности для S-блоков малой битности (преимущественно черехбитовых и меньше). Рассмотрены классы алгебраических преобразований, не изменяющих вид DDT. Разработан алгоритм восстановления специального класса эквивалентности по конкретной DDT.

S-БЛОК, ТАБЛИЦА РАСПРЕДЕЛЕНИЯ ДИФФЕРЕНЦИАЛОВ, КЛАСС DDT-ЭКВИВАЛЕНТНОСТИ, ВОССТАНОВЛЕНИЕ S-БЛОКА ПО DDT, МОДУЛЬНОЕ СЛОЖЕНИЕ

ABSTRACT

This diploma work contains: 80 pages, 1 drawing, 5 tables, 9 sources.

The purpose is the development of new approaches and methods for analyzing cryptographic primitives, which will create new reliable cryptographic information security systems and assess the stability of existing ones.

The object of research information processes in information security systems.

subject of study are algebraic properties of cryptographic nonlinear transformations.

In this paper we consider the problem of recovering S-blocks from the differential distribution table (hereinafter DDT) in respect to modular conditions. The structure of special classes of equivalence for S-blocks of small bitness (mainly four-bit and lower) is investigated. The classes of algebraic transformations that do not change the kind of DDT are considered. An algorithm for the reconstruction of a special class of equivalence for a specific DDT has been developed.

S-BLOCK, DIFFERENTIAL DISTRIBUTION TABLE, DDT-EQUIVALENT CLASS, S-BLOCK RECOVERING FROM DDT, MODULAR ADDITION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

$\oplus$ — операція побітового додавання

$\mathbb{F}_2^n$ — скінчене поле (поле Галуа) характеристики 2 зі степенем розширення n

$\langle a, b \rangle$ — скалярний добуток векторів a та b

DDT — таблиця розподілу диференціалів (англ. Difference Distribution Table)

LAT — таблиця лінійних апроксимацій (англ. Liniar Approximation Table)

APN — майже повністю нелінійні (англ. Almost Perfect Nonliniar)

SAC — строгий лавиний ефект (англ. Strict Avalence Criterion)

ВСТУП

Актуальність дослідження. Актуальність даного дослідження полягає в тому, що в наш час відкриті канали зв'язку, такі як, мережа Internet, посідають вагоме місце в життєдіяльності людства. Людство все більше покладається на відкриті мережі з метою спілкування чи проведення тих чи інших операцій. Проте слід пам'ятати, вразі передачі даних відкритими мережами зловмисник може перехопити конфіденційні дані, що може завдати шкоди. Питання забезпечення цілісності та конфіденційності інформації є все більш хвилюючим, тому висуваються все жорсткіші стандарти забезпечення захисту інформації. Одним з широко розповсюджених математичних напрямків забезпечення інформаційної безпеки є симетрична криптографія.

Стійкість блочних шифрів зумовлена наявністю в їх структурі S-блоків, які роблять нелінійне обратиме перетворення над вхідними бітами, що перешкоджає проведенню лінійного криптоаналізу.

Проте S-блоки є вразливими до іншого виду криптоаналізу - диференційного. Пошук протидії атакам, які базуються на досягненнях диференційного криптоаналізу, дав поштовх до вивчення S-блоків та їх властивостей. Важливою складовою в оцінці диференційних характеристик S-блоків є таблиця розподілу диференціалів (англ. Difference Distribution Table) (далі DDT). DDT легко обчислити, маючи S-блок, однак обернена задача з відновлювання S-блока за DDT є складною. Відновлення S-блока за DDT має важливий практичний зміст, оскільки DDT містить змістовний опис диференційних властивостей S-блока, таким чином можемо отримати S-блок з наперед заданими властивостями за таблицею розподілу диференціалів.

Метою дослідження є розробка нових підходів та методів до аналізу криптографічних примітивів, що дозволить створювати нові надійні системи криптографічного захисту інформації та оцінювати

стійкість існуючих. Для досягнення мети потрібно розв'язати наступні задачі:

1. провести огляд опублікованих джерел за тематикою дослідження;

2. сформувати та довести аналітичні властивості таблиць розподілу диференціалів відносно модульного додавання;

3. сформувати та проаналізувати задачу відновлення S-блоку за таблицею розподілу диференціалів;

4. запропонувати алгоритм відновлення на основі підходу Backtracking та експериментально його перевірити.

Об'єктом дослідження є інформаційні процеси в системах криптографічного захисту.

Предметом дослідження є алгебраїчні властивості криптографічних нелінійних перетворень.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: теорії імовірностей, математичної статистики, комбінаторного аналізу, теорії кодування, теорії складності алгоритмів, методи комп'ютерного та статистичного моделювання.

Наукова новизна у роботі вперше розглядалась задача відновлення криптографічного S-блоку за таблицею розподілу диференціалів відносно додавання за модулем степеня двійки та запропановано алгоритм її розв'язання, який є практично ефективний для S-блоків невеликого розміру.

Практичне значення. Результати даної роботи можуть бути використані у задачах пошуку прихованих аналітичних структур криптопримітивів та при оцінюванні стійкості криптографічних систем у моделі «білої скриньки».

Апробація результатів та публикації. Результати даної роботи частково були представленні в рамках XVII Науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики» (26-27 квітня 2019р., м. Київ).

1 КЛЮЧОВІ ОЗНАЧЕННЯ ТА ОГЛЯД ЛІТЕРАТУРИ

У даному розділі розглянуто базові означення з теорії булевих функцій, основні криптографічні властивості булевих функцій, засоби їх дослідження. Приділено увагу ключовій для криптографії сутності, а саме S-блокам, описано методи їх дослідження; окрема увага акцентована на суттєвих інструментах дослідження криптографічних властивостей S-блоків: DDT та LAT. Серед іншого розглянуто афінну еквівалентність S-блоків та її зв'язок з відновленням S-блока за DDT у випадку побітового додавання. Розглянута проблема відновлення S-блока за DDT в контексті диференціалів за побітовим додаванням. На завершення розглянуто роботу, присвячену розв'язанню проблеми відновлення S-блока у випадку традиційних для диференціального криптоаналізу диференціалах – на основі додавання за модулем двійки.

1.1 Необхідні теоретичні відомості

1.1.1 Булеві функції та їх криптографічні властивості

В сучасній критології природною математичною моделлю S-блоків виступають багатовимірні булеві функції. Тому перш ніж досліджувати S-блоки, розглянемо булеві функції та їх основні криптографічні властивості.

Зауваження. Протягом роботи простір бітових векторів розмірності n позначається через $\mathbb{F}_2^n$, який залежно від контексту може інтерпретуватися як кільце $\mathbb{Z}_{2^n}$; у цьому випадку двійкові вектори природним чином ототожнюються з цілими невід'ємними числами, двійковий запис яких співпадає із заданим вектором.

Означення 1.1. Одновимірною булевою функцією $n - f$ з n вхідними значеннями називається відображення виду $\mathbb{F}_2^n \to \mathbb{F}_2$.

Означення 1.2. m-вимірною булевою функцією $(n, m) - F$ з n вхідними значення та m вихідними значеннями будемо називати відображення виду $\mathbb{F}_2^n \to \mathbb{F}_2^m$.

Зауваження. В цій роботі позначення $(n, m) - F$ використовується, щоб підкреслити, коли багатовимірна булева функція F має n вхідних бітів та m вихідних; для одновимірних має сенс зазначати лише кількість вхідних бітів, оскільки на виході завжди маємо один біт.

Незважаючи на те, що для опису S-блоків застосовуються саме багатовимірні булеві функції, в цьому розділі в основному описано властивості булевих функцій на прикладі одновимірних функцій з уточненнями для багатовимірного випадку, адже в багатьох випадках аналіз багатовимірних булевих функцій зводиться до аналізу одновимірних булевих функцій. Оскільки для узагальнення теоретичних результатів завжди можна представити m-вимірну булеву функцію як кортеж з m одновимірних булевих функцій: $F = (f_1, f_2, \ldots, f_m)$. Ці функції називаються *координатними функціями.*

Булеві функції можуть бути представлені у вигляді таблиці істинності, АНФ, ДНФ або спектрально. Розглянемо найбільш зручні для аналітичних досліджень форми представлення булевих функцій.

Означення 1.3. Алгебраїчною нормальною формою (АНФ) булевої функції f називається її представлення у вигляді полінома Жегалкіна:

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u x^{(u)}, \quad a_u \in \{0, 1\},$$

де $x^{(u)}$ - піднесення до степеня, що визначається наступним чином:

$$x^{(u)} = \begin{cases} x, \text{ якщо } u = 1, \\ 1, \text{ якщо } u = 0. \end{cases}$$

Теорема 1.1. *(Мебіус) Коефіцієнти АНФ можуть бути обчислені через швидке перетворення Мебіуса:*

$$a_u = \bigoplus_{x \preceq u} f(x).$$

Серед усіх представлень булевих функцій слід виокремити найбільш зручні для аналітичного дослідження – спектральні розклади. Розглянемо *розклад Фур'є* булевої функції.

Означення 1.4. Розкладом Фур'є одновимірної булевої функції $f(x)$ називається представлення булевої функції у наступному вигляді:

$$f(x) = \sum_{u \in \mathbb{F}_2^n} C_f(u)(-1)^{\langle u,x \rangle},$$

де $\sum_{u \in \mathbb{F}_2^n}$ – арифметична сума над $\mathbb{Z}$,
$\langle u, x \rangle$ – скалярний добуток векторів, що обчислюється наступним чином:
$\langle u, x \rangle = u_1 x_1 \oplus u_2 x_2 \oplus \cdots \oplus u_n x_n$, де n – розмірність векторів u та x,
$C_f(u)$ – коефіцієнт Фур'є, що визначається наступною формулою:

$$C_f(u) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} f(x)(-1)^{\langle u,x \rangle}$$

Проте на практиці доволі часто використовують інші коефіцієнти для аналізу властивостей булевих функцій, які є більш зручними.

Означення 1.5. Коефіцієнтом Уолша (Уолша-Адамара) називається величина, що визначається наступним чином:

$$W_f(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle x,u \rangle \oplus f(x)}.$$

Зауваження. Будемо називати функції виду $g(x) = (-1)^{f(x)}$ як ± 1 аналогом функції $f(x)$. Таким чином, наведені коефіцієнти Уолша є коефіцієнтами Фур'є ± 1 аналога функції $f(x)$.

Чисельні характеристики коефіцієнтів Уолша є дуже потужним

інструментом для дослідження криптографічних характеристик булевих функцій, оскільки багато властивостей тим чи іншим чином пов'язані з коефіцієнтами Уолша.

Означення 1.6. Вагою булевої функції f називається кількість входів, на яких значення виходу булевої функції дорівнює 1, тобто

$$\mathrm{wt}(f) = |\{x \in \mathbb{F}_2^n | f(x) = 1\}|.$$

Означення 1.7. Алгебраїчним степенем булевої функції f називається величина, що визначена як

$$\deg f = \arg\max_{u \in \mathbb{F}_2^n} \{\mathrm{wt}(u), a_u = 1\}.$$

Зауваження. Для багатовимірних булевих функцій алгебраїчний степінь дорівнює максимальному алгебраїчному степені координатних функцій.

Існує певний набір властивостей, що ускладнюють чи навіть унеможливлюють криптоаналіз S-блоку, що побудований на основі булевої функції, та можуть слугувати критеріями для вибору булевої функції для криптографічних цілей. Розглянемо основні важливі для криптографії властивості булевих функцій.

Невиродженість

Означення 1.8. Змінна x_k називається неістотною для булевої функції $(n, m) - F$, якщо виконується рівність

$$\forall a \in \mathbb{F}_2^n \quad F(a \oplus e_k) = F(a), \text{ де } e_k \text{ – вектор, що має 1 на } k\text{-ому місці.}$$

Означення 1.9. Булева функція називається невиродженою, якщо всі її змінні є істотними.

З точки зору криптографії доволі важливо, щоб булева функція не була вироджена на будь-яких вхідних даних. Адже вироджена функція

може бути спрощеною на певних вхідних даних, що в свою чергу значно спрощує складність шифру в цілому, оскільки ця змінна ніяк не впливає на вихід булевої функції. Враховуючи цей фактор, зловмисник може здійснити повний перебір зі значно меншою складністю.

Збалансованість

Означення 1.10. Булева функція $(n, m) - F$ називається збалансованою, якщо кожне значення функції досягається рівну кількість разів:

$$\forall y \in \mathbb{F}_2^m : |\{x \in \mathbb{F}_2^n | F(x) = y\}| = 2^{n-m}.$$

З теорії інформації відомо, що рівноймовірний ансамбль має найбільшу можливу ентропію. Рівноймовірность на множині вихідних значень досягається у випадку, коли частоти кожного можливого значення функції рівні, тобто в тому випадку, якщо функція є збалансованою. Збалансовані булеві функції мають дуже гарні криптографічні властивості, проте через їх малу кількість доволі часто розглядають майже збалансовані булеві функції.

Нелінійність

Серед вимог, що висунуто криптографією до булевих функцій - кандидатів на основу шифру - є висока нелінійність, адже лінійні перетворення над вхідними даними є доволі передбачуваними і нестійкими для багатьох атак.

Означення 1.11. $n-f$ називається афінною булевою функцією, якщо її алгебраїчний степінь менший або рівний за одиницю, тобто $\deg f \leq 1$.

Лінійні та афінні функції є дуже слабкими з точки зору криптографії, оскільки їх значення дуже швидко та легко обчислюються та є доволі передбачуваними завдяки апарату лінійної алгебри та чисельних методів.

Означення 1.12. Булева функція $n - f$ виду $l_a(x) = \langle a, x \rangle$ називається афінною булевою функцією, $a \in \mathbb{F}_2^n$.

Твердження 1.1. *Нехай $\mathbb{A}_n$ - множина усіх афінних булевих функцій, тоді $|\mathbb{A}_n| = 2^{n+1}$.*

Як бачимо, кількість булевих функцій є незначною відносно кількості всіх булевих функцій, проте доволі часто нелінійні булеві функції ефективно апроксимуються лінійними аналогами, що дуже спрощує шифр у цілому.

Введемо наступну метрику для відображення ступеня нелінійності.

Означення 1.13. Відстанню Гемінга між булевими функціями f та g будемо називати величину виду

$$\mathrm{dist}(f, g) = |\{x \in \mathbb{F}_2^n | f(x) \neq g(x)\}| = \mathrm{wt}(f \oplus g).$$

З огляду на це можна більш строго формалізувати нелінійність булевої функції.

Означення 1.14. Нелінійністю булевої функції f називається

$$NL_f = \mathrm{dist}(f, \mathbb{A}_n) = \max_{l \in \mathbb{A}_n} \mathrm{dist}(f, l).$$

Твердження 1.2. *Нелінійність булевої функції може бути обчислена наступним чином:*

$$NL_f = 2^{n+1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |W_f(a)|.$$

Якщо спроектувати означення нелінійності на багатовимірний випадок, то маємо наступне означення.

Означення 1.15. Нехай $(n, m) - F$ багатовимірна булева функція, тоді її нелінійність визначається як $NL_f = \min_{\beta \in \mathbb{F}_2^n} NL_{\langle \beta F \rangle}$, де $\langle \beta F \rangle$ є лінійною комбінацією координатних функцій F.

Наостанок введемо важливий для криптографії клас булевих функцій.

Означення 1.16. Булева функція f називається *бент-функцією*, якщо $\forall a \in \mathbb{F}_2^n \quad W_f(a) = 2^{\frac{n}{2}}$.

Кореляційний імунітет

Для криптографії дуже важливо, щоб значення на виході булевої функції містило якомога менше інформації про значення на вході. Ця властивість є важливою, якщо є необхідність, щоб шифр був «цілком таємним за Шеноном». Адже чим менше інформації у виході шифру, тим менше інформації про вхід має зловмисник, маючи вихідні значення. Щоб формалізувати це твердження, введемо властивість *кореляційного імунітету.*

Нехай $f : \mathbb{F}_2^n \to \{0, 1\}$ та на $\mathbb{F}_2^n$ заданий рівномірний розподіл. Тоді введемо наступну випадкову величину

$$Y = f(x), \quad Pr\{Y = 1\} = \frac{\mathrm{wt}(f)}{2^n}.$$

Означення 1.17. Функція f має кореляційний імунітет рівня k $(1 \leq k \leq n)$ $\mathrm{cor} f = k$, якщо виконується одна з наступних умов:

1) $\forall i_1, i_2, \ldots, i_k \subseteq \{1, 2, \ldots, n\} : I(Y_i, x_{i_1}, x_{i_2}, \ldots, x_{i_k}) = 0,$

де взаємна інформація визначається через ентропію наступним чином: $I(X, Y) = H(X) + H(Y) - H(X, Y)$;

2) кожна $(n - k)$-підфункція f', одержана фіксацією довільних k змінних довільними значеннями, має вагу $\mathrm{wt}(f') = \frac{\mathrm{wt}(f)}{2^k}$.

Зауваження. Наведені умови є еквівалентними.

Слід розуміти, що про кореляційний імунітет доречно говорити, якщо на множині вхідних значень задано імовірнісний розподіл.

Властивість кореляційного імунітету дозволяє протидіяти класу *кореляційних атак*, коли криптоаналітик може використовувати кореляційні залежності між бітами входу та виходу та таким чином може отримати окремі біти входу.

Означення 1.18. Булева функція f-кореляційно імунна за напрямком $a \in \mathbb{F}_2^n$, якщо $c(f(x), l_a(x)) = 0$, де $c(f, g)$ - коефіцієнт

кореляції булевих функцій f, g, що може бути обчислений наступним чином:

$$c(f,g) = \frac{1}{2^n}(|\{x \in \mathbb{F}_2^n | f(x) = g(x)\}| - |\{x \in \mathbb{F}_2^n | f(x) \neq g(x)\}|).$$

Твердження 1.3. *$corf = k \Leftrightarrow f$ кореляційно імунна за усіма a, де $1 \leq wt(a) \leq k$.*

Існує наступне співвідношення між коефіцієнтом кореляції та коефіцієнтами Уолша

$$c(f,g) = \frac{1}{2^n} W_f(a).$$

Враховуючи це, можемо переписати останнє твердження наступним чином.

Твердження 1.4. $corf = k \Leftrightarrow \forall\ a \quad 1 \leq wt(a) \leq k \quad W_f(a) = 0.$

Лавинний ефект

Перейдемо до опису наступних властивостей, а саме: *лавинних ефектів та критеріїв поширення.* Лавинний ефект (англ. Avalance Effect) - це вплив зміни вхідного біту на результуюче значення функції. Чим більш непередбачувані та великі зміни у виході відбуваються через зміну одного вхідного біту, тим краще з точки зору криптографії, адже навіть при більш-менш схожих вхідних даних, будуть отримані різні виходи.

Означення 1.19. Булева функція f задовольняє строгому лавинному ефекту (англ. Strict Avalence Criterion або SAC) за змінною x_i тоді і тільки тоді, коли зміна x_i призводить до зміни значення функції рівно у половині випадків.

Означення 1.20. Введемо коефіцієнт розповсюдження помилки як величину, що має наступний вид

$$k_i = \sum_{x \in \mathbb{F}_2^n} (f(x) \oplus e_i)).$$

Отже, з огляду на введене означення можна помітити, що булева функція задовольняє SAC за $x_i \Leftrightarrow k_i = 2^{n-1}$.

Формалізуємо поняття лавинного ефекту для багатовимірних булевих функцій.

Твердження 1.5. *$(n, m) - F$ задовольняє SAC рівня k_i за x_i, якщо* $\forall j_1, j_2, \ldots, j_{k+1} \subseteq \{1, \ldots, m\} \quad f_{j_1} \oplus f_{j_2} \oplus \ldots \oplus f_{j_{k+1}}$ *задовольняє SAC за x_i.*

Наведене означення є доволі строгим, тому на практиці доволі часто розглядають дещо послаблену вимогу.

Твердження 1.6. *$(n, m) - F$ задовольняє SAC за x_i в середньому, якщо* $k_i(F) = \sum\limits_{x \in \mathbb{F}_2^n} wt(F(x) \oplus F(x \oplus e_i)) = 2^{n+m-1}$.

Підсумовуючи вище викладене, можна сказати, що функції, що мають строгий лавинний ефект, мають властивість, яка зумовлена тим, що зміна одного вхідного біту призводить до непередбачуваних змін бітів на вході. Таким чином, це є дуже важливим при протидії диференційним криптоатакам, що побудовані на спостереженні змін вихідних значень при зміні вхідних значень. Крім цього, ця властивість є дуже важливою при побудові якісної геш-функції та генераторів псевдовипадкових чисел.

До цього моменту було розглянуто вплив однієї змінною на вихід булевої функції, але частою практичною задачею є дослідження впливу декількох змінних на вихід. Саме для цього вводять наступний математичний апарат.

Означення 1.21. Похідною булевої функції за напрямком $a \in \mathbb{F}_2^n$ називається величина, що визначена наступним чином:

$$D_a f(x) = f(x \oplus a) \oplus f(x).$$

Твердження 1.7. *Булева функція f задовольняє критерію поширення за напрямком $a \in \mathbb{F}_2^n$, якщо $wt(D_a f) = 2^{n-1}$.*

Зауваження. SAC за $x_i \Leftrightarrow$ критерій поширення за напрямком e_i.

Твердження 1.8. *Булева функція f задовольняє критерію поширення рівня k $(1 \leq k \leq n)$, якщо f задовольняє критерію поширення за $\forall\ a$ $(1 \leq wt(a) \leq k)$.*

Слід зауважити, що лавинний ефект та кореляційний імунітет є основними властивостями, що протидіють атакам за обраними відкритими текстами.

Означення 1.22. Функція взаємної кореляції булевих функцій f, g - це дійснозначна функція виду

$$\triangle_{f,g}(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus g(x \oplus a)}.$$

Зауваження. Коефіцієнт кореляції є частковим випадком кореляційної функції $c(f, g) = \triangle_{f,g}(0)$.

Означення 1.23. Функція автокореляції

$$\triangle_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus f(x \oplus a)}.$$

Твердження 1.9. *Функція автокореляції є перетворенням Фур'є квадратів коефіцієнтів Уолша*

$$\triangle_f(a) = \frac{1}{2^n} \sum_{u \in \mathbb{F}_2^n} (-1)^{au} W_f^2(u),$$

$$W_f^2(u) = \sum_{a \in \mathbb{F}_2^n} (-1)^{au} \triangle_f(a).$$

Слід зауважити, що усі наведені властивості пов'язані між собою, і тому неможливо отримати функцію, що має всі властивості на високому рівні.

Всі теоретичні відомості стосовно криптографічних властивостей булевих функцій, що наведені в цьому розділі, були наведені, як результат опрацювання відповідних джерел, а саме [1, 2, 3].

1.1.2 S-блоки та їх властивості

Багато криптографічних примітивів, таких як, наприклад, блочні шифри, використовують багатовимірні булеві функції як свою структурну одиницю, які прийнято називати *Substitution-блоками* (далі *S-блоки*). Для блочних шифрів S-блоки відіграють дуже важливу роль, оскільки вони перемішують вхідні значення та роблять шифр стійким до чисельних криптографічних атак. Зокрема, добре підібрані S-блоки гарантують стійкість шифру до лінійного та диференціального криптоаналізу, а також до ряду алгебраїчних атак.

Як зазначалось, для опису структури S-блока доволі часто використовують координатне представлення у вигляді багатовимірної булевої функції, проте існує ще одна математична модель для представлення S-блоків – перестановки без повторень. Оскільки представлення у вигляді багатовимірної булевої функції є зручнішим для аналітичного дослідження, тому якщо не сказано іншого, в цій роботі використовується саме цей спосіб представлення. Таким чином, S-блок, що є відображенням виду $\mathbb{F}_2^n \to \mathbb{F}_2^m$, може бути інтерпретований, як багатовимірна булева функція $(n, m) - F$, яка в свою чергу може бути представлена у вигляді m координатних булевих функцій з n змінними.

Наведемо приклад такого представлення у вигляді таблиці 1.1.

Таблиця 1.1 – Приклад 4-бітового S-блока

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	f	e	b	c	6	d	7	8	0	3	9	a	4	2	1	5
$S_1(x)$	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1
$S_2(x)$	1	1	1	0	1	0	1	0	0	1	0	1	0	1	0	0
$S_3(x)$	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	1
$S_4(x)$	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0

Наразі, можна побачити, що S-блок $S(x)$ є $(4,4) - F$ булевою функцією, що може бути представлена через координатні булеві функції $(S_1(x), S_2(x), S_3(x), S_4(x))$.

Завдяки швидкому перетворенню Мебіуса може бути доволі ефективно обчислено АНФ координатних функцій:

$$S_1 = 1 + x_1 + x_3 + x_2x_3 + x_4 + x_2x_4 + x_3x_4 + x_1x_3x_4 + x_2x_3x_4$$
$$S_2 = 1 + x_1x_2 + x_1x_3 + x_1x_2x_3 + x_4 + x_1x_4 + x_1x_2x_4 + x_1x_3x_4$$
$$S_3 = 1 + x_2 + x_1x_2 + x_2x_3 + x_4 + x_2x_4 + x_1x_2x_4 + x_3x_4 + x_1x_3x_4$$
$$S_4 = 1 + x_3 + x_1x_3 + x_4 + x_2x_4 + x_3x_4 + x_1x_3x_4 + x_2x_3x_4.$$

Слід зауважити те, що координатні функції та їх лінійні комбінації повністю визначають усі криптографічні властивості S-блока. Тому аналіз криптографічних властивостей S-блока може бути розглянутий в термінах вищезазначених криптографічних властивостей булевих функцій. З урахуванням чого теоретичні результати для булевих функцій можуть бути застосовані при дослідженні S-блоків.

1.1.3 Методи та інструменти дослідження S-блоків

На сьогоднішній день своє почесне місце в інструментарії сучасного криптоаналізу займає широко розповсюджений напрям, винайдений Елі Біхамом та Аді Шаміром, який прийнято називати *диференційним криптоаналізом*. Цей напрям є доволі потужним та породив цілу течію у криптографії, сприявши виникненню ефективних підходів для побудови криптоатак на шифри, що є стійкими до засобів *лінійного криптоаналізу*. Основною ідеєю диференційного криптоаналізу є дослідження різниці між вихідними значеннями шифру на різних вхідних значеннях та її еволюції при проходженні через різні раунди шифру. Висновки на основі

цих спостережень дають змогу зробити припущення щодо структури та властивостей S-блока або навіть щодо найбільш вірогідних бітів ключа.

В рамках диференційного криптоаналізу було розроблено багато методів та інструментів для дослідження криптографічних характеристик S-блоків. Зокрема, стійкість до диференціального криптоаналізу визначається *таблицею диференціальних імовірностей* (англ. Difference Distribution Table, або просто DDT), що містить значення імовірностей усіх диференціалів S-блоку.

Означення 1.24. The difference distribution table (DDT) (таблиця розподілу диференціалів) S-блоку — двовимірна таблиця, кожна комірка якої містить кількість таких входів, для яких пари з різницею a переходить у пару виходів із різницею b.

Тобто для вхідної різниці $a \in \mathbb{F}_2^n$ та вихідної різниці $b \in \mathbb{F}_2^m$ комірка $\delta(a, b)$ DDT має наступний вигляд:

$$\delta(a, b) = |\{z \in \mathbb{F}_2^n | S(z \oplus a) \oplus S(z) = b\}|.$$

Зауваження. Наведене вище означення DDT може бути переписано через похідну: $\forall a, b \in \mathbb{F}_2^n \quad \delta(a, b) = |\{x \in \mathbb{F}_2^n | D_a F(x) = b\}|$.

Зауваження. В деяких джерелах використовується термін *«таблиця диференціальних імовірностей»*, оскільки DDT фактично є таблицею чисельників імовірностей усіх диференціалів S-блоку, відповідно імовірності диференціалів відповідно визначаються як $\delta(a, b) \cdot 2^{-n}$.

Існує інший різновид DDT, що побудований на основі додавання за модулем степеня двійки. Тобто в цьому випадку комірка має наступний вигляд:

$$\delta(a, b) = |\{z \in \mathbb{F}_2^n | S(z + a) = b + S(z)\}|,$$

де $+ -$ додавання за $mod\ 2^n$.

Останній наведений вид DDT, що побудований на основі диференціалів за модулем 2^n, є доволі важливим, адже для окремих

блокових шифрів (таких як сімейство SAFER чи алгоритм шифрування ДСТУ ГОСТ 28147:2009), саме такі диференціали розглядати більш природно, аніж диференціали за побітовим додаванням.

Зауваження. Якщо не вказано іншого в цьому розділі під DDT, маються на увазі звичайні DDT з додаванням за модулем два.

Для дослідження та порівняння між собою DDT в спеціалізованій літературі можна побачити наступний інструмент – *індикатор DDT*.

Означення 1.25. Індикатором DDT називається булева функція, визначена наступним чином:

$$\gamma_F(a,b) = 0 \iff \left[\begin{array}{l} \delta_F(a,b) = 0, \\ a = 0. \end{array}\right.$$

Доволі часто водночас з DDT розглядають ще один розповсюджений інструмент для дослідження S-блоків – *таблицю лінійних апроксимацій (англ. Linear Approximation Table або LAT)*, яка використовується для отримання наближеного лінійного співвідношення між вхідними бітами та вихідними бітами S-блока. Незважаючи на те, що LAT є центральним інструментом у *лініному криптоаналізі*, існує вже добре вивчений зв'язок між DDT та LAT.

Означення 1.26. The linear approximation table (таблиця лінійних апроксимацій) S-блоку – двомірна таблиця, в якій для входу $a \in \mathbb{F}_2^n$ та виходу $b \in \mathbb{F}_2^m$ кожна комірка має наступне значення:

$$\lambda(a,b) = |\{x \in \mathbb{F}_2^n | ax \oplus bS(x) = 0\}| - 2^{n-1} = \frac{1}{2}\sum_{x \in \mathbb{F}_2^n} (-1)^{ax \oplus bS(x)}.$$

Важливим є той факт, що відновлення S-блоку за LAT є тривіальним, адже маючи LAT певного S-блока, зловмисник може з легкістю відтворити цей самий S-блок, розв'язавши систему лінійних рівнянь (Лема 2 [7]).

Зауваження. Доволі часто запис LAT визначають як узагальнення

коефіцієнта Уолша, в цьому ключі вони приймають вид:

$$W_F(a,b) = \sum_{x \in \mathbb{F}_2^n} (-1)^{ax+bF(x)}.$$

Підкреслимо, що основні інструменти дослідження S-блоків, а саме DDT та LAT, базуються на інструментах дослідження булевих функцій, таких як похідна булевої функції за напрямком та коефіцієнти Уолша.

Означення 1.27. Лінійністю булевих функції називається максимальний коефіцієнт у LAT при $b \neq 0$.

Зауважимо, що з огляду на якість S-блоку з точки зори криптографічних властивостей слід мінімізувати лінійність булевої функції задля підвищення стійкості шифру до лінійних атак, що є доволі потужним класом атак на симетричні криптосистеми.

1.1.4 Класи еквівалентності S-блоків

Однією з основних складностей, що виникає при відновленні S-блока за його DDT, є той факт, що існують випадки, коли різні S-блоки мають однакову DDT чи доволі схожі за тією чи іншою ознакою, що значно ускладнює поставлену задачу. Тому на практиці вводять *класи еквівалентності S-блоків*.

Означення 1.28. S-блоки $S_1(x)$ та $S_2(x)$ називаються DDT-еквівалентними, якщо вони мають однакові DDT.

Означення 1.29. S-блоки $S_1(x)$ та $S_2(x)$ називаються γ-еквівалентними, якщо індикатори їх DDT рівні: $\gamma_{S_1} = \gamma_{S_2}$; іноді цю еквівалентність також називають диференційною еквівалентністю.

Зауваження. Очевидно, що з DDT-еквівалентності слідує γ-еквівалентність S-блоків, проте обернене твердження не завжди

справедливе.

Множина S-блоків, що є попарно DDT-еквівалентними, утворює так званий *клас DDT-еквівалентності*. Аналогічно S-блоки, що є попарно γ-еквівалентними, утворюють *клас γ-еквівалентності*.

Твердження 1.10. *(Властивість 1 [7]) Нехай $S(x)$ є $n \times m$ S-блоком та позначимо l розмірність його лінійного простору, тобто простору утвореного всіма можливими лінійними перетвореннями $S(x)$. Тоді клас DDT-еквівалентності $S(x)$ налічує 2^{n+m-l} різних функцій виду:*

$$x \to S(x \oplus c) \oplus d, \ \textit{де } c \in \mathbb{F}_2^n, \ d \in \mathbb{F}_2^m.$$

Означення 1.30. Нехай $S_1(x)$ та $S_2(x)$ S-блоки, що є булевими функціями виду $\mathbb{F}_2^n \to \mathbb{F}_2^n$, тоді вони є афінно-еквівалентними, якщо існують афінні перетворення A_1, A_2, такі, що виконується наступне співвідношення:

$$\exists \ A_1, A_2 : \mathbb{F}_2^n \to \mathbb{F}_2^n \quad S_1(x) = A_2(S_2(A_1(x))).$$

Означення 1.31. Нехай $S_1(x)$ та $S_2(x)$ S-блоки, тоді вони є EA-еквівалентними, якщо існує представлення виду:

$$S_1(x) = A_2(S_2(A_1(x))) \oplus A_0,$$

де A_0, A_1, A_2 - афінні функції $\mathbb{F}_2^n \to \mathbb{F}_2^n$, причому на A_1 та A_2 накладається умова бієктивності.

Зауваження. В цій роботі, якщо не вказано іншого, розглядаються афінні перетворення в умовах тієї ж операції, в якій визначені диференціали.

Отже, на практиці не існує бієктивного відображення із множини S-блоків у множину DDT, що значно ускладнює задачу відновлення S-блока за DDT, оскільки S-блоки, що зв'язані афінним перетворенням, утворюють клас еквівалентності. Тому з практичної точки зору зручно

мати алгоритми, що дають змогу отримати весь клас по одному представнику або перевіряти, чи є два S-блоки афінно-еквівалентними.

Означення 1.32. Диференціальною однорідністю S-блока $S(x)$ називається $\max\limits_{a\in\mathbb{F}_2^n\setminus\{0^n\},b\in\mathbb{F}_2^m}\delta(a,b)$.

Твердження 1.11. *Найменше можливе значення диференціальної однорідності функції з* $(n,n)-F$ *дорівнює 2.*

Означення 1.33. Функція, що має найменше можливе значення диференційної однорідності називається *майже ідеально нелінійною (англ. Almost Perfect Nonlinear або APN).*

APN функції забезпечують надійний захист проти диференційних криптоатак. Існує багато добре досліджених скінчених класів APN функцій. Слід мати на увазі, що відновлення APN S-блока за DDT є складнішим. Наприклад, доволі відомі визначені над скінченим полем $\mathbb{F}_2^n$, так звані, *функції Голда (the Gold functions)*, що мають наступний вигляд x^{2^i+1}, де на i накладається наступна умова $gcd(i,n)=1$ для всіх непарних $n>1$. Зауважимо, що питання існування таких функцій для парних n є відкритим.

Щоб підкреслити змістовність DDT зазначимо, що навіть її вигляд може дати багато інформації при оцінці стійкості до різноманітних криптоатак. Не тільки значення максимальної диференційної однорідності, але і розташування в межах DDT може свідчити про підвищення або погіршення стійкості до тієї чи іншої атаки.

Серед іншого для формування теоретичних відомостей стосовно диференійного та лінійного криптоаналізу та пов'язаних питань опрацьовано роботу [4].

1.2 Задача відновлення S-блока за таблицею розподілу диференціалів

1.2.1 Постановка задачі та проблематика

Задача відновлення S-блока за DDT є фактично дискретним інтегруванням та може розглядатись аналогічно до задачі інтегрування в непервному випадку. Так, аналогічно до задачі інтегрування з початковими умовами в неперервному випадку, для відновлення конкретного S-блока потрібні додаткові початкові умови, а саме значення S-блока в певних точках.

Проте за аналогією до інтегрування, не маючи ніяких початкових умов, окрім самої DDT, можна вирішити задачу в загальному випадку, тобто відновивши цілий клас DDT-еквівалентності.

Розглянемо практичний зміст даної задачі. На сьогоднішній день одним з розповсюджених підходів пошуку S-блоку із якісними криптографічними характеристиками полягає у випадковому генеруванні S-блоку (з урахуванням певних визначених обмежень), після чого досліджуються криптографічні параметри згенерованого S-блоку, зокрема, структура DDT. Подивимось на цей процес з іншої сторони. Так, отримання S-блоку, що має задану DDT, є важливою практичною задачею з декількох причин.

По-перше, такий метод дає змогу отримати S-блок з наперед заданими властивостями, що суттєво краще для вибору S-блоку. Таким чином, маючи алгоритм відновлення S-блока за DDT, можна спочатку згенерувати DDT, що описує S-блок, який задовольняє основним потребам, після чого на основі згенерованої DDT можна отримати власне S-блок.

По-друге, для класу криптографічних атак у моделі, так званої «білої скриньки», зловмисник може отримати DDT невідомого йому S-блоку та,

маючи в своєму арсеналі алгоритм для відновлення, він може отримати сам S-блок, що значно полегшує злам шифру. Відповідно, аналіз методів відновлення дозволяє оцінити стійкість криптопримітивів у моделі «білої скриньки».

1.2.2 Аналіз опублікованих результатів

Незважаючи на вагомі перспективи, відновлення S-блока за DDT ще не є дослідженим до кінця. В цьому розділі буде наведений огляд зроблених досліджень по відновленню S-блока з наперед заданою DDT.

Слід зазначити, що існують декілька алгоритмів відновлення S-блока, що має конкретну DDT. Деякі з них є доволі ефективними, деякі мають ефективність не кращу за звичайний перебір. Розглянемо декілька вже існуючих алгоритмів.

Виокремимо роботу [5], автори якої, Орр Дункельман та Сен'янь Хуань, запропонували новий алгоритм відновлення, який ґрунтується на співвідношенні між DDT та LAT S-блоку, дослідженому у [9], [7], [8]. Цей алгоритм працює краще, ніж запропонований раніше алгоритм Guess-and-Determine, представлений у [6].

Сформуємо зазначений вище зв'язок між DDT та LAT. Для початку наведемо наступне перетворення над функціями.

Означення 1.34. Нехай $f : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{R}$, тоді функція f' називається перетворенням Уолша-Адамара:

$$\forall (a,b) \ \in \mathbb{F}_2^n \times \mathbb{F}_2^m \quad f'(a,b) = \sum_{x,y} f(x,y)(-1)^{ax \oplus by},$$

де x, y пробігають всі цілі числа.

Теорема 1.2. *(Зв'язок DDT та LAT)* $\forall\ (a,b) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$

$$\begin{cases} \delta'(a,b) = 4\lambda^2(a,b), \\ 4\lambda^{2\prime}(a,b) = 2^{m+n}\delta(a,b), \end{cases}$$

де $\lambda^{2\prime}(a,b)$ *є перетворенням Уолша-Адамара* $\lambda^2(a,b)$.

Як можна побачити з першого рівняння, у теоремі 1.2, квадрат елементу LAT лінійно пропорційний до перетворення Уолша-Адамара елемента DDT. З урахуванням чого сам S-блок може бути отриманий з LAT через перетворення Уолша-Адамара. Таким чином, задача з відновлення S-блока за DDT зводиться до задачі встановлення знаку елементу LAT. В контексті роботи [5] ця задача називається *задача відновлення знаку (англ. The Sign Determination Problem).*

Зауваження. В цьому підрозділі, якщо не вказано іншого, будуть розглядатися S-блоки як функції виду $\mathbb{F}_2^n \to \mathbb{F}_2^m$.

Опишемо деякі позначення, що будуть використовуватися в ході подальших викладок. Нехай позначимо LAT як λ, тоді вектор-стовбець $\overline{\lambda}_b$ є b-тою колонкою LAT. Також введемо наступне позначення $\lambda^\dagger(a,b)$, що означає абсолютне значення $\lambda(a,b)$ – елемента LAT. На останок позначимо вектор $((-1)^{bS(0)}, \ldots, (-1)^{bS(2^n-1)})$ через $\overline{s}_b$.

Таким чином, LAT $\lambda^\dagger$, що складається з абсолютних значень, може бути записана наступним чином: $(\overline{\lambda}_0^\dagger, \ldots, \overline{\lambda}_{2^n-1}^\dagger)$.

Отже, маючи в своєму розпорядженні DDT, завдяки співвідношенню наведеному у теоремі 1.2 можна з легкістю отримати будь-яку комірку LAT $\lambda^\dagger(a,b)$. Тобто, щоб отримати всі значення LAT λ, необхідно відновити знак для кожної комірки $\lambda^\dagger(a,b)$.

Означення 1.35. Нехай відомі абсолютні значення b-того стовпчика LAT $\overline{\lambda}_b^\dagger$, тоді *задача відновлення знаку* для b-того стовпчика LAT полягає в отриманні знакових значень $\overline{\lambda}_b$ з абсолютних значень $\overline{\lambda}_b^\dagger$, де $0 \le b < 2^m$.

Твердження 1.12. *(Властивість 2, [5]) Існує наступне*

твердження стосовно b-того стопчика LAT

$$\forall \quad 0 \leq b < 2^m \quad H_n \overline{s}_b = 2\overline{\lambda}_b,$$

де H_n *- матриця Адамара порядка* 2^n*.*

Будемо називати колонки LAT $\lambda_0, \dots \lambda_j$ *незалежними колонками*, якщо $\lambda_0, \dots \lambda_j$ є лінійно незалежними над $\mathbb{F}_2^m$, де $0 \leq j < m$.

Слід розуміти, що якщо зловмисник вирішить задачу відновлення знака для m незалежних колонок, тоді S-блок може бути відновлений завдяки розв'язку системи лінійних рівнянь зі складністю $O(2^n m^3)$. Звісно, зловмисник може вирішити задачу відновлення знака, використовуючи повний перебір, але в цьому випадку складність асимптотично дорівнює $O(2^{m2^n})$, так як слід перебрати m колонок з 2^n елементами у кожній.

Так, в роботі [5] було запропоновано застосувати рекурсивний алгоритм для відновлення знаків елементів LAT. В ході роботи алгоритму на кожному кроці рекурсії розв'язується система лінійних рівнянь, що є вдвічі меншою порівняно з попереднім кроком. Та на n-му кроці маємо вирішення нашої задачі відновлення знака.

Опишемо основну ідею даного алгоритму та проілюструємо його роботу на прикладі.

Алгоритм приймає на вхід абсолютні значення стовпця LAT $\overline{\lambda}_b^{\dagger}$ та дає на виході множину можливих значень векторів-стовпців LAT $\overline{\lambda}_b$, що можуть дати ці абсолютні значення.

Основною математичною ідеєю, що покладена до фундаменту цього алгоритму є наступна властивість матриці Адамара (Властивість 3 [5]):

$$H_i = \begin{bmatrix} H_{i-1} & H_{i-1} \\ H_{i-1} & -H_{i-1} \end{bmatrix}, i \geq 1.$$

Використовуючи (Властивість 2 [5]) та (Властивість 3 [5]) отримаємо

наступні результати (деталі отримання наведені у розділі 3.2.1 [5]):

$$H_{n-l}\overline{s}_b^{[0,2^{n-l}-1]} = \overline{\gamma}_0, H_{n-l}\overline{s}_b^{[2^{n-l},2^{n-l+1}-1]} = \overline{\gamma}_1$$
$$\vdots$$
$$H_{n-l}\overline{s}_b^{[2^n-2^{n-l+1},2^n-2^{n-l}-1]} = \overline{\gamma}_{2^l-2}, H_{n-l}\overline{s}_b^{[2^n-2^{n-l},2^n-1]} = \overline{\gamma}_{2^l-1},$$

де

$$\overline{\gamma}_0 = \big(\overline{\beta}_0^{[0,2^{n-l}-1]} + \overline{\beta}_0^{[2^{n-l},2^{n-l+1}-1]}\big)/2, \overline{\gamma}_1 = \big(\overline{\beta}_0^{[0,2^{n-l}-1]} - \overline{\beta}_0^{[2^{n-l},2^{n-l+1}-1]}\big)/2$$
$$\vdots$$
$$\overline{\gamma}_{2^l-2} = \big(\overline{\beta}_{2^{l-1}-1}^{[0,2^{n-l}-1]} + \overline{\beta}_{2^{l-1}-1}^{[2^{n-l},2^{n-l+1}-1]}\big)/2, \overline{\gamma}_{2^l-1} = \big(\overline{\beta}_{2^{l-1}-1}^{[0,2^{n-l}-1]} - \overline{\beta}_{2^{l-1}-1}^{[2^{n-l},2^{n-l+1}-1]}\big)/2.$$

Та $\overline{\beta}_0, \ldots, \overline{\beta}_{2^{l-1}-1}$ - вектори, отримані на попередньому рівні рекурсії l. Зазначимо, що l пробігає значення $1 \leq l \leq n$. Покажемо роботу алгоритму на наступному прикладі:

$$\overline{\lambda}_b^{\dagger} = (1,1,1,1),\ \overline{\lambda}_b = (1,1,1,-1),\ \overline{s}_b = (1,1,1,-1).$$

Позначимо l-ий рівень рекурсії як T_l. Проініціалізуємо початкові значення, як $T_0[i] = \{2\lambda_b[i], -2\lambda_b[i]\}$, де $0 \leq i \leq 2^n$. Отже, маємо $T_0[0] = T_0[1] = T_0[2] = T_0[3] = \{2, -2\}$. Після чого можемо обчислити кожний рівень рекурсії $T_l[i]$ за наступною формулою:

$$((p_0 + q_0)/2, (p_0 - q_0)/2, (p_{2^{l-1}-1} + q_{2^{l-1}-1})/2, (p_{2^{l-1}-1} - q_{2^{l-1}-1})/2),$$

де $\overline{p} \in T_{l-1}[i],\ \overline{q} \in T_{l-1}[i + 2^{n-l}]$. Таким чином, маємо таке заповнення для наступного рівня рекурсії

$$T_1[0] = T_1[1] = \{(2,0), (0,2), (0,2), (-2,0)\}.$$

Слід зазначити, що область допустимих значень є {1, -1}. Отже, на

останньому рівні рекурсії при $p = (2, 0), q = (2, 0)$ маємо наступний вектор:

$$E_1(p, q) = ((2 + 2)/2, (2 - 2)/2, (0 + 0)/2, (0 - 0)/2) = (2, 0, 0, 0).$$

Через те, що $2 \notin \{1, -1\}$, цей вектор не включається до $T_2[0]$. Отже, на виході алгоритму отримаємо наступну множину:

$$T_2[0] = \{(1, 1, 1, -1), (-1, -1, -1, 1), (1, 1, -1, 1), (-1, -1, 1, -1),$$
$$(1, -1, 1, 1), (-1, 1, -1, -1), (-1, 1, 1, 1), (1, -1, -1, -1)\}.$$

Як можна побачити, $\overline{s}_b \in T_2[0]$. Виконання алгоритму може бути зображено у вигляді деревовидної структури як на малюнку **??**:

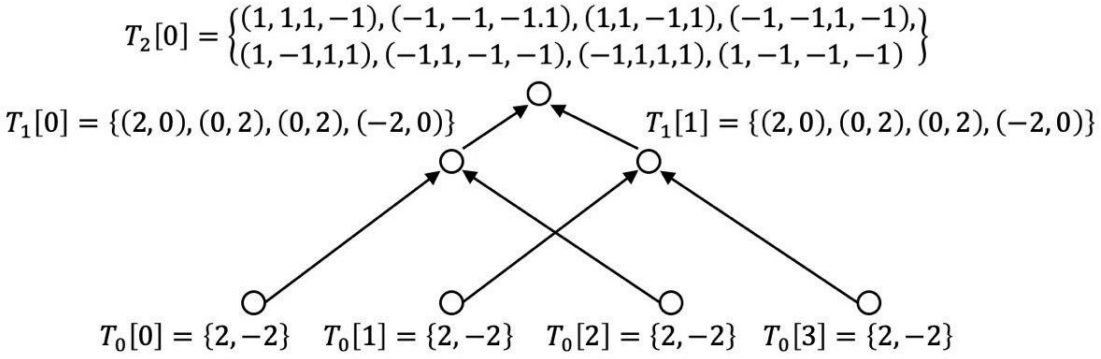

Рисунок 1.1 – Приклад роботи алгоритму у древовидній формі.

Також в рамках цієї роботи була представлена оптимізована з точки зору складностей по часу та по пам'яті версія цього алгоритму. Основною ідеєю оптимізації було вдосконалення алгоритму, в результаті якого більше немає необхідності зберігати в пам'яті множину всіх векторів поточного рівня рекурсії, а лише множину, що є їх компактним представленням.

В роботі [6] був представлений алгоритм, що приймає на вхід $2^n \times 2^n$ таблицю DDT, заповнену невід'ємними цілими числами, та повертає всі функції F виду $\mathbb{F}_2^n \to \mathbb{F}_2^n$, DDT яких мають такий же індикатор γ_D, як і індикатором таблиці D. Слід зазначити, що фільтруючи клас γ-еквівалентності можна отримати клас DDT еквівалентності.

Також в роботі [6] було піднято питання, чи існують два S-блоки F та G, що є DDT-еквівалентними та не пов'язані співвідношенням $G(x) = F(x \oplus c) \oplus d$ для деяких c, d. Серед іншого було встановлено, що у випадку не бієктивних відображень такі пари існують.

Висновки до розділу 1

Отже, в першому розділі цієї роботи описано теоретичний базис, на який спираються наші дослідження, а саме: розглянуто основні означення математичної теорії булевої функції та розглянуто основні криптографічні властивості булевих функцій, на які слід звернути увагу для вибору функції для реалізації S-блоку.

Розглянуто S-блоки та їх основні криптографічні властивості, а саме принципи представлення S-блоків, та приділено основну увагу представленню у вигляді багатовимірних булевих функцій. Розглянуто основні інструменти для дослідження властивостей S-блоків, а саме DDT та LAT. Розглянуто основні важливі для поставленої задачі класи еквівалентності S-блоків.

Проведено аналіз опублікованих джерел на досліджувану тему. Детальну увагу було приділено роботі [5], а саме запропонованому в цій роботі алгоритму для вирішення задачі відновлення S-блока за заданою DDT у випадку диференціалів, побудованих за побітовим додаванням. Встановлено, що, якщо розглядати цю задачу у випадку диференціалів при додаванні за модулем 2^n, розглянуті вище співвідношення між DDT та LAT, на яких базуються результати роботи [5], не є дійсними у загальному випадку. На сьогодні для випадку модульного додавання немає аналітичних результатів, тому наступний розділ присвячений саме цій проблемі.

2 ЗАДАЧА ВІДНОВЛЕННЯ S-БЛОКІВ ЗА ТАБЛИЦЕЮ РОЗПОДІЛУ ДИФЕРЕНЦІАЛІВ У ВИПАДКУ МОДУЛЬНОГО ДОДАВАННЯ

В другому розділі науково-дослідницької роботи представлено власні дослідження стосовно питань, що відносяться до проблеми відновлення S-блока за таблицею розподілу диференціалів у випадку диференціалів, побудованих відповідно до додавання за модулем степеня двійки (далі модульне додавання). Зауважимо, що, якщо не вказано іншого, в цьому розділі всі питання розглядаються у контексті модульного додавання. Так, в другому розділі описано вид та структуру DDT лінійних S-блоків. Розглянуто афінні перетворення та особливу увагу приділено їхньому впливу на вид DDT. Описано деякі корисні з практичної точки зору властивості DDT. Частково досліджено структуру класів *DDT-еквівалентності*. Запропоновано алгоритм відновлення частини класу DDT-еквівалентності, базуючись на проведених дослідженнях.

2.1 Властивості таблиці розподілу диференціалів у випадку модульного додавання

2.1.1 Лінійні S-блоки

Розпочнемо опис структури класів DDT-еквівалентності з розгляду класу найпримітивніших S-блоків, а саме *лінійних*. Нехай позначення $S(x)$ буде означати n-бітовий бієктивний S-блок. Так, емпіричним шляхом було

встановлено, що S-блоки виду

$$S(x) = ux + w, \text{ де } u \in \mathbb{Z}_{2^n}^*, w \in \mathbb{Z}_2^n$$

мають DDT характерного виду. Зауважимо, що для дотримання умови бієктивності S-блока, ми накладаємо умову $u \in \mathbb{Z}_{2^n}^*$.

Розглянемо більш детально елемент DDT лінійного S-блока $\delta(a, b)$:

$$\begin{aligned}\delta(a,b) = |\{x \in \mathbb{F}_2^n \mid S(x+a) = S(x) + b\}| = \\ |\{x \in \mathbb{F}_2^n \mid ux + au + w = ux + w + b\}| = \\ |\{x \in \mathbb{F}_2^n \mid au = b\}|.\end{aligned}$$

Аналізуючи останній вираз можна помітити, що значення диференціалу au не залежить від змінної x. Таким чином, з цього випливає, що елементи DDT лінійних S-блоків $\delta(a, b)$ мають обмежену область допустимих значень, а саме $\delta(a, b) \in \{0, 2^n\}$. При чому з виду виразу $au = b$ та його незалежності від x можна помітити, що в рядку DDT a існує лише один ненульовий елемент, що розміщений на позиції au та дорівнює 2^n, а всі інші – 0.

Зазначимо те, що окрім можливих значень елементів DDT лінійних S-блоків існують ще умови на позиції елемента 2^n в рамках рядка.

Розглянемо непарні рядки, тобто $a = 2k + 1, k \in \mathbb{N}$. Виходячи з умови $u \in \mathbb{Z}_{2^n}^*$, маємо, що $u = 2k + 1$. Як результат, можна помітити, що:

$$b = au = (2k + 1)(2k + 1) = 4(k^2 + k) + 1,$$

тобто ненульовий елемент 2^n може бути тільки на непарних позиціях в непарних рядках.

В випадку парних рядків ($a = 2k, k \in \mathbb{N}$) зауважимо, що, виходячи з

$$b = au = 2k(2k + 1) = 2(2k^2 + k),$$

елемент 2^n може бути розміщений тільки на парних позиціях. Але слід

брати до уваги те, що у цьому випадку задля існування коренів рівняння $au = b$ накладається додаткова умова на коефіцієнти $a,\ b$, а саме: добутки коефіцієнтів $agcd^{-1}(a,b)$ та $bgcd^{-1}(a,b)$ повинні належати $\mathbb{Z}_{2^n}^*$, де $gcd(a,b)$ – найбільший спільний дільник a та b.

Серед усього іншого було помічено, що на вигляд DDT S-блоку впливає лише коефіцієнт u, тобто при фіксованому значенні u всі S-блоки мають еквівалентні DDT, що є частковим випадком більш глобального співвідношення, яке буде наведено далі.

Зазначимо, що, виходячи з наведеного вище факту, кількість унікальних класів DDT-еквівалентності лінійних S-блоків дорівнює:

$$\phi(2^n) = 2^{n-1},$$

де $\phi(x)$ – функція Ойлера, n – бітова розрядність S-блока.

При чому кількість лінійних n-бітових S-блоків може бути оцінена наступним виразом:

$$2^n\phi(2^n) = 2^{2n-1}.$$

Так, з цього слідує, що потужність кожного класу DDT-еквівалентності лінійних S-блоків становить 2^n.

2.1.2 Афінні перетворення

Розглянемо афінні перетворення у випадку модульного додавання та їхній вплив на вигляд DDT.

В ході аналізу предметної області було встановлено декілька співвідношень між DDT, S-блоки яких пов'язані афінними перетвореннями аргументів та виходів.

Формалізуємо ці результати у вигляді наступних теорем.

Теорема 2.1. *$S_1(x)$ та $S_2(x)$ – n-бітові S-блоки, що пов'язані афінним перетворенням аргументів, тобто*

$$\exists\ u, w \in \mathbb{F}_2^n \quad S_1(x) = S_2(ux + w),$$

тоді справедливе наступне співвідношення для їх DDT

$$\forall\ a, b \in \mathbb{F}_2^n \quad \delta_{S_1}(a, b) = \delta_{S_2}(ua, b).$$

Доведення.

$$\delta_{S_1}(a, b) = |\{x \in \mathbb{F}_2^n \mid S_1(x + a) = S_1(x) + b\}| =$$
$$|\{x \in \mathbb{F}_2^n \mid S_2(ux + ua + w) = S_2(ux + w) + b\}|$$

Введемо заміну $y = ux + w$:

$$|\{y \in \mathbb{F}_2^n \mid S_2(y + ua) = S_2(y) + b\}| = \delta_{S_2}(ua, b),$$

що й треба було довести. □

Отже, у випадку афінного перетворення вхідних аргументів DDT S-блоків однакові з точністю до перестановки рядків.

Теорема 2.2. *$S_1(x)$ та $S_2(x)$ – n-бітові S-блоки, пов'язані афінним перетворенням виходів, тобто*

$$\exists\ u \in \mathbb{Z}_{2^n}^* \quad \exists\ w \in \mathbb{F}_2^n \quad S_1(x) = uS_2(x) + w,$$

тоді має місце наступне співвідношення між їх DDT:

$$\forall\ a, b \in \mathbb{F}_2^n \quad \delta_{S_1}(a, b) = \delta_{S_2}(a, u^{-1}b).$$

Доведення.

$$\delta_{S_1}(a,b) = |\{x \in \mathbb{F}_2^n \mid S_1(x+a) = S_1(x) + b\}| =$$
$$|\{x \in \mathbb{F}_2^n \mid uS_2(x+a) + w = uS_2(x) + b + w\}| =$$
$$|\{x \in \mathbb{F}_2^n \mid uS_2(x+a) = uS_2(x) + b\}|$$

Оскільки $u \in \mathbb{Z}_{2^n}^*$, то існує u^{-1}, а тому:

$$|\{x \in \mathbb{F}_2^n \mid S_2(x+a) = S_2(x) + u^{-1}b\}| = \delta_{S_2}(a, u^{-1}b),$$

що й треба було довести. □

Отже, у випадку афінного перетворення виходів DDT S-блоків однакові з точністю до перестановки рядків.

З практичної точки зору для нас важливі такі афінні перетворення, які зберігають DDT. З теорем 2.1 та 2.2 випливає, що афінні перетворення виду

$$S(x) \to S(x+a) + b,$$

де $a, b \in \mathbb{F}_2^n$, не змінюють DDT S-блоку. Будемо говорити, що таке перетворення утворює *клас афінних зсувів*, а саме перетворення будемо називати *афінним зсувом*.

Однак експериментальним шляхом було встановлено, що не тільки S-блоки, що пов'язані афінними зсувами, є DDT-еквівалентними. Сформуємо це спостереження у вигляді наступної теореми:

Теорема 2.3. *$S_1(x)$ та $S_2(x)$ – n-бітові S-блоки, що пов'язані родиною афінних перетворень виду*

$$S_1(x) = -S_2(-x + u) + w,$$

тоді S_1 та S_2 мають однакові DDT, тобто

$$\forall\ a, b \in \mathbb{F}_2^n \quad \delta s_1(a,b) = \delta s_2(-a,-b),$$

де від'ємні константи природним чином інтерпретуються наступним чином $-a = 2^n - a$.

Доведення.

$$\delta_{s_1}(a,b) = |\{x \in \mathbb{F}_2^n \mid S_1(x+a) = S_1(x) + b\}| =$$
$$|\{x \in \mathbb{F}_2^n \mid -S_2(-x+u-a) + w = -S_2(-x+u) + w + b\}| =$$
$$|\{x \in \mathbb{F}_2^n \mid S_2(-x+u-a) = S_2(-x+u) + -b\}|$$

Введемо заміну $y = -x + u$:

$$|\{y \in \mathbb{F}_2^n \mid S_2(y-a) = S_2(y) + -b\}| = \delta_{s_2}(-a,-b),$$

що й треба було довести. □

Будемо називати родину афінних перетворень, що розглядаються в теоремі 2.3, *оберненими афінними зсувами*.

А клас, що згенерований комбінаціями обернених та звичайних афінних зсувів аргументів та виходів, – *класом афінних зсувів еквівалентності*.

Зауважимо, що з теореми 2.3 випливає декілька цікавих фактів. По-перше, обернені афінні зсуви дійсно утворюють S-блоки з рівними DDT. По-друге, матриця DDT без нульового стовпця та нульової строки є симетричною відносно додаткової діагоналі.

Таким чином, не тільки афінні зсуви зберігають DDT: існують такі афінні перетворення, що являють собою перестановки рядків і стовців DDT, в результаті яких вид DDT залишається незмінним.

Слід зазначити, що на практиці існують більш складні трансформації, ніж афінні перетворення, що зберігають вид DDT. Ці залежності можна простежити в S-блоках великої бітності, що, на жаль, не є дослідженими у цій роботі.

Наведемо аналітичні оцінки потужностей розглянутих вище класів. Для порівняння зауважимо, що всього існує $n!$ різних n-бітових S-блоків.

Так, можна помітити, що для довільного n-бітового ($n \geq 3$) S-блоку

$S(x)$ кількість нетрівіальних афінно-еквівалентних йому S-блоків, що мають таку ж саму DDT з точністю до перестановки рядків та стовпців, дорівнює

$$(2^n)^2\phi^2(2^n) = \frac{2^{4n}}{4} = 2^{4n-2},$$

де $\phi(n)$– функція Ойлера.

Та для довільного n-бітового ($n \geq 3$) S-блоку $S(x)$ кількість нетрівіальних афінно-еквівалентних S-блоків, що входять до класу афінних зсувів, дорівнює 2^{2n}.

Зауважимо, що клас обернених афінних зсувів має таку ж саму потужність, як і клас афінних зсувів. А клас афінних зсувів еквівалентності налічує 2^{2n+1} S-блоків.

Для наглядності, наскільки великими на практиці можуть бути класи афінної еквівалентності, наведемо числові значення потужності класів еквівалентності від розмірності S-блоку.

Результати розрахунків для класів афінної еквівалентності наведено в таблиці 2.1, де $|x|$ — розрядність S-блока у бітах, $\#S(x)$ – кількість S-блоків.

Таблиця 2.1 – Залежність потужності класу афінної еквівалентності від бітової розрядності S-блока

$\|x\|$	3	4	5	6	7	8
$\#S(x)$	1024	16384	262144	4194304	67108864	1073741824

Результати розрахунків потужностей класів афінних зсувів наведено у таблиці 3.1, що має такі самі позначення.

Таблиця 2.2 – Залежність потужності класу афінних зсувів від бітової розрядності S-блока

$\|x\|$	3	4	5	6	7	8
$\#S(x)$	64	256	1024	4096	16384	65536

2.1.3 Відомості про структуру таблиць розподілу диференціалів та класів DDT-еквівалентності

В ході виконання науково-дослідницької роботи було встановлено декілька корисних фактів щодо структури таблиць розподілу диференціалів та класів DDT-еквівалентності, що є базисом для представленого в цій роботі алгоритму відновлення.

Твердження 2.1. *Для довільного n-бітового S-блоку $S(x)$ перший рядок його DDT має наступний вигляд:*

$$(2^n, 0, 0, \ldots, 0).$$

Доведення. Нехай $S(x)$ – довільний n-бітовий S-блок, тоді розглянемо його похідну за напрямком a:

$$D_a S(x) = S(x + a) - S(x).$$

Покладемо $a = 0$, тоді розподіл цієї похідної формує перший рядок DDT:

$$S(x + 0) - S(x) = 0.$$

Так, похідна за цим напрямком є тотожним нулем для $\forall x \in \mathbb{Z}_{2^n}$.

Отже, запис DDT $\delta(0,0) = 2^n$, а $\delta(0,i) = 0, 1 \leq i \leq n-1$.

Що й треба було довести. □

Серед іншого було помічено, що не для кожного S-блоку усі афінні зсуви та обернені афінні зсуви утворюють унікальні S-блоки. Тобто класи афінних зсувів еквівалентності мають різні потужності в залежності від виду S-блоку.

Твердження 2.2. *Для довільного n-бітового лінійного S-блоку $S(x)$ в результаті застосування усіх можливих афінних зсувів та обернених афінних зсувів можна отримати n унікальних S-блоків.*

Доведення. Нехай $S(x)$ довільний n-бітовий лінійний S-блок, що має наступний вигляд $S(x) = ax + b$, де коефіцієнт $a \in \mathbb{Z}_{2^n}^*$ та коефіцієнт $b \in \mathbb{Z}_{2^n}$.

Розглянемо S-блок $S'(x)$, що пов'язанний з $S(x)$ комбінацією двох довільних афінних зсувів виходів та аргументів, тоді:

$$S'(x) = S(x+u) + w = a(x+u) + b + w = ax + au + b + w = ax + z,$$

де $z = au + b + w$.

Очевидно, що $z \in \mathbb{Z}_{2^n}$. Так, для лінійних S-блоків комбінація довільних двох афінних зсувів аргументів та виходів еквівалентна одному афінному зсуву виходів. Отже, з властивостей афінних зсувів S-блок $S'(x)$ належить тому ж самому класу DDT-еквівалентності, що й $S(x)$.

Зрозуміло, що для лінійних S-блоків клас афінних зсувів виходів та входів має потужність не більшу, ніж клас афінних зсувів виходів. Зазначимо, що кількість унікальних лінійних S-блоків, що можливо отримати таким чином залежить від кількості унікальних значень z, що обмежено значенням n.

Нехай тепер S-блок $S'(x)$, що пов'язаний з $S(x)$ комбінацією двох

довільних обернених афінних зсувів виходів та аргументів, тоді:

$$S'(x) = (n-1)S((n-1)x+u)+w = (n-1)(a((n-1)x+u)+b)+w = \\ (n-1)(anx-ax+au+b)+w = -anx+ax-au-b+w = \\ ax-au-b+w = ax+z,$$

де $z = -au-b+w$.

Проводячи аналогічні висновки можна отримати, що довільною комбінацією обернених афінних зсувів входу та виходу отримуємо не більше ніж n унікальних S-блоків.

Що й треба було довести. □

Так, можна помітити, що не всі класи DDT-еквівалентності мають однакові потужності.

Твердження 2.3. *На кожному рядку DDT довільна фіксована точка S-блока x може попасти в одну й тільки одну з комірок рядка DDT, причому на іншому рядку ця ж точка може попасти в іншу комірку.*

Доведення. За означенням елементу DDT маємо:

$$\delta_s(a,b) = |\{x \in \mathbb{F}_2^n \mid S(x+a) = S(x)+b\}| = \\ |\{x \in \mathbb{F}_2^n \mid S(x+a)-S(x) = b\}|.$$

Проаналізуємо останній вираз, а саме частину $S(x+a)-S(x)=b$.

Можна помітити, що при фіксованому рядку DDT a для кожної точки S-блока x позиція елементу DDT b в рамках одного рядка, куди потрапляє ця точка, є строго визначеною та залежить лише від x. З цього випливає перша половина твердження.

Враховуючи те, що рядки DDT є незалежними та для кожного іншого рядка змінюється лише a, то з цього випливає друга половина твердження.

Що й треба було довести. □

Так, було отримано підтвердження того, що кожна точка S-блоку на кожному рядку є на фіксованій позиції.

Твердження 2.4. *Нехай $S_1(x)$ - довільний n-бітовий S-блок, який належить до класу DDT-еквівалентності $\mathbb{G}$, тоді $\exists$ $S_2(x)$ такий, що $S_2(0) = 0$ та $S_2(x) \in \mathbb{G}$.*

Доведення. Нехай $S_1(x)$ має значення a ($a \in \mathbb{F}_2^n$) в точці 0, тобто $S_1(0) = a$. Тоді розглянемо S-блок $S_2(x)$, який пов'язаний з $S_1(x)$ наступним афінним зсувом:

$$S_2(x) = S_1(x) - a.$$

З вигляду $S_2(x)$ очевидно, що $S_2(0) = 0$. Також раніше було встановлено, що афінні зсуви не змінюють DDT S-блока. А це означає, що $S_1(x), S_2(x) \in \mathbb{G}$, де $\mathbb{G}$ – клас DDT-еквівалентності S-блоків.

Отже, $S_2(x)$ і є шуканий S-блок. □

Таким чином, було встановлено, що в кожному класі DDT-еквівалентності існує хоча б один представник, такий що $S(0) = 0$.

2.2 Алгоритм відновлення S-блока за таблицею розподілу диференціалів

В цьому підрозділі розглянемо алгоритм відновлення класу афінних зсувів еквівалентності. Зауважимо, що у випадку, що розглядається в постановці задачі по відновленню, немає жодної додаткової інформації про відновлюваний S-блок, окрім DDT.

В попередньому розділі було досліджено структуру та властивості класів DDT-еквівалентності. Таким чином, враховуючи наведені вище результати, для розв'язання задачі відновлення класу DDT-еквівалентності можна знайти лише одного представника класу DDT-еквівалентності з заданою DDT, що далі буде називатися твірним елементом, та на його основі породити клас афінних зсувів

еквівалентності, що для S-блоків з бітністю 3 та менше співпадає з класом DDT-еквівалентності.

Основною ідеєю даного алгоритму є покрокове висунення припущень щодо значення S-блока в певній точці та перевірка коректності припущення. Базуючись на результатах перевірки, можна прийняти рішення, чи вгадувати значення S-блока в наступній точці, чи перегадувати в поточній. Таким чином, можна отримати значно кращі результати, аніж при наївному переборі всіх можливих варіантів, що прийнято називати *повним перебором* (або англ. *Brute Force*), адже можна відкинути цілі множини S-блоків, які точно не мають задану DDT. Даний підхід не є інноваційним та є відомим в англомовній літературі як *Backtracking*.

Для реалізації алгоритму на основі Backtracking необхідно обрати правило, на основі якого можна визначати незадовільні варіанти та відкидати їх.

Задля зручності та оптимізації в силу справедливості твердження 2.4 покладемо $S[0] = 0$. Тоді можна висувати припущення щодо елементів S-блоку $S(x)$, починаючи з $a = 1$.

Нехай далі будемо позначати множину ненульових елементів DDT для a-того рядку як $\mathbb{A}_a$. Розглянемо значення похідної S-блоку $S(x)$ за напрямком a, яка має вигляд $S(x - a) - S(x)$. Знаючи вигляд DDT S-блока $S(x)$, можна зробити висновок, що можливі значення похідної обмежені значеннями ненульових елементів a-го рядка DDT $\mathbb{A}_a$. Якщо ж розглядати значення похідної в точці 0, то вона приймає наступний вигляд $S(a)$, що фактично є розташуванням точки 0 S-блоку на довільному рядку DDT a. Тоді з огляду на твердження 2.3, про однозначну відповідність позиції в рамках рядка DDT та точки S-блоку, можна організувати перебір у нашому Backtracking алгоритмі наступним чином: спочатку робимо припущення щодо значення виразу $S(a)$, після чого перевіряється можливість даного припущення, що можна організувати за наступним правилом. Після фіксування 0 на рядку a

$(a \geq 1)$ перевіряти наступне твердження для всіх k $(0 < k < a)$:

$$\delta(a, S(a) - S(k)) = \begin{cases} 0, \text{припущення невірне,} \\ \text{інакше, припущення може бути вірним.} \end{cases}$$

Зауважимо, що можливі значення k є $\overline{1, a-1}$ та отримані з наступних міркувань. Перевірка значення 0 не має значення, оскільки, виходячи з припущення, що $S(0) = 0$, в цьому випадку фактично перевіряється $\delta(a, S(a) \neq 0)$, що є завжди справедливо з огляду стратегії вибору елементу $S(a)$.

Тобто, якщо для всіх k наведений елемент DDT має ненульове значення, вважаємо припущення про розташування 0 можливим і переходимо до наступного рядка DDT $a + 1$. Якщо ж хоча б для одного k існує елемент DDT $\delta(a, S(a) - S(k))$ рівний нулеві, то вважаємо такий варіант хибним та перевгадуємо розташування 0.

Нехай будемо позначати множину перевірених варіантів значень $S(a)$ для a-того рядку як $\mathbb{T}_a$. Тоді в разі $\mathbb{A}_a = \mathbb{T}_a$ слід повернутися на крок вище та обнулити множину перевірених варіантів.

Наведемо псевдокод рекурсивної процедури, що є вгадуванням елементу S-блоку та повертає представника класу DDT еквівалентності. З псевдокодом даної процедури можна ознайомитись в додатку 3.2.

Маючи процедуру вгадування, представимо псевдокод алгоритму для відновлення твірного S-блоку з заданою DDT у випадку n-бітового S-блоку:

Algorithm 2.1 Алгоритм відновлення твірного n-бітового S-блоку з заданою DDT

Вхід: DDT ddt, для якої шукаємо твірний елемент

Вихід: Твірний S-блок $S(x)$

цикл *для всіх цілих i від 0 до n-1* **виконати**
| $S(i) = 0$

кінець циклу

цикл *для всіх* $\delta_i j$ *у DDT ddt* **виконати**

| **якщо** $\delta_i j \neq 0$ **тоді**
| | додати j до множини $\mathbb{A}_i$
| **кінець умови**

кінець циклу

цикл *для всіх цілих i від 0 до n-1* **виконати**
| $\mathbb{T}_i = \varnothing$

кінець циклу

викликати процедуру вгадування **??** з параметрами $\{1; \{\mathbb{A}_a : 0 \leq a \leq n\}; \{\mathbb{T}_a : 0 \leq a \leq n\}; \text{ddt}\}$;

Наведемо модифікацію з точки зору пам'яті для запропонованого алгоритму. Так, можна значно скоротити кількість кроків та перевірок і не використовувати додаткову пам'ять на запам'ятовування множин перевірених варіантів $\mathbb{T}_a$, адже в умовах обмежених обчислювальних ресурсів це може значно погіршувати роботу алгоритму. Тоді запропонована процедура вгадування буде мати наступний вигляд:

Algorithm 2.2 Оптимізована версія алгоритму відновлення класу афінних зсувів еквівалентності

Вхід: Позиція S-блока a, що вгадується;

Родина множин ненульових елементів рядків DDT $\{\mathbb{A}_a : 0 \leq a \leq n\}$;

Позиція $offset$, з якої починається перебір поточного рівня;

Таблиця розподілу диференціалів ddt;

Вихід: Один із твірних елемнтів класу афінних зсувів, що має DDT ddt

цикл *для всіх цілих i від* $offset$ *до* $n-1$ **виконати**

- **якщо** $\mathbb{A}_\ni[i]$ *немає серед вгаданих елементів S-блоку* $S(x)$ **тоді**
  - $S[a] = \mathbb{A}_\ni[i]$ **якщо** *a = n-1* **тоді**
    - **якщо** *S-блок* $S(x)$ *має DDT ddt* **тоді**
      - **повернути** *S(x)*
    - **кінець умови**
  - **кінець умови**
  - **цикл** *для всіх цілих x від 0 до a* **виконати**
    - **якщо** $\delta(a, S(a) - S(k)) = 0$ **тоді**
      - рекурсивно викликати процедуру з параметрами $\{a+1;\ \{\mathbb{A}_a : 0 \leq a \leq n\};\ 0;\ \text{ddt}\}$;
    - **кінець умови**
  - **кінець циклу**
- **кінець умови**

кінець циклу

S(a) = 0 і = бінарний пошук $S[a-1]$ у $\mathbb{A}_a$ рекурсивно викликати процедуру з параметрами $\{a+1;\ \{\mathbb{A}_a : 0 \leq a \leq n\};\ i;\ \text{ddt}\}$;

Зауважимо, що для пошуку індексу елементу $S[a-1]$ в масиві $\mathbb{A}_a$ для часової оптимізації можна використовувати бінарний пошук, адже $\mathbb{A}_a$ є відсортованими через характер ініціалізації.

Для наглядності проілюструємо роботу описаного алгоритму на коротенькому прикладі у випадку DDT 2-ух бітового S-блоку $(0, 2, 3, 1)$,

що має DDT виду:

$$\begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 1 & 2 & 1 \end{pmatrix}$$

Отже, множини ненульових елементів DDT та множини перевірених варіантів будуть проініціалізовані наступним чином:

$$\mathbb{A}_0 = \{0\}, \mathbb{A}_1 = \{1,2,3\},\ \mathbb{A}_2 = \{1,3\},\ \mathbb{A}_3 = \{1,2,3\};$$
$$\mathbb{T}_0 = \varnothing,\ \mathbb{T}_1 = \varnothing,\ \mathbb{T}_2 = \varnothing,\ \mathbb{T}_3 = \varnothing.$$

Початкове заповнення S-блоку є $(0,0,0,0)$. Нульовий елемент не чіпається, адже він фіксується нулем. На першому кроці покладемо на першу позицію S-блока перший елемент з множини можливих елементів для цієї позиції $\mathbb{A}_1$. Отже, $S[1] = 1$, тоді множина спроб матиме вигляд $\mathbb{T}_1 = \{1\}$. На наступному кроці маємо 2 можливих варіанта з множини $\mathbb{A}_2$, проте, оскільки 1 вже стоїть на першій позиції S-блоку, покладемо $s[2] = 3$ та $\mathbb{T}_2 = \{1,3\}$. Так, на третьому кроці не можна взяти 1, тому покладемо $S[3] = 2$, тоді множина спроб $\mathbb{T}_3 = \{1,2\}$. S-блок $(0,1,3,2)$ має шукану DDT. Алгоритм завершує свою роботу.

На останок наведемо певні оцінки часової складності даного алгоритму. Слід зазначити, що верхня асимптотична оцінка складності алгоритму залишається такою ж самою, як і в випадку повного перебору, тобто $O(n!)$. Проте в запронованому алгоритмі наявні деякі оптимізації, що роблять алгоритм значно швидшим. Якщо брати до уваги той факт, що перший елемент S-блоку фіксується нулем, то зазначимо, що виконується перебір на $(n-1)!$ варіантів менше. Якщо на a-ому кроці правило говорить, що подальший перебір не має сенсу, то в цьому

випадку необхідно перебрати на $(n-a)!$ варіантів менше.

Зазначимо, що у випадку лінійних S-блоків алгоритм завершує роботу майже миттєво через специфічну структуру таблиць розподілу диференціалів лінійних S-блоків, оскільки для кожного елементу S-блока S(a) існує всього один можливий варіант.

Висновки до розділу 2

Отже, в цьому розділі розглянуто властивості та описано структуру таблиць розподілу диференціалів для лінійних S-блоків у випадку, коли диференціали обчислюються за модулем степеня двійки. Встановлено, що лінійні S-блоки мають легко впізнаваємі DDT завдяки обмеженню на значення їх елементів та розташування елементів в рамках рядків DDT. Серед іншого наведено оцінки кількості лінійних S-блоків, унікальних DDT та потужностей класів DDT-еквівалентності.

Розглянуто основні властивості афінних перетворень в контексті таблиць розподілу диференціалів. Показано, що певні класи афінних перетворень над $\mathbb{Z}_2^n$ зберігають DDT, зокрема, класи афінних зсувів та обернених афінних зсувів аргументів та виходів. Також виведені формули для обрахунку потужностей кожного важливого для практики класу та наведені результати обрахунків для S-блоків малих бітностей.

Описано певні важливі для практики властивості DDT для довільних S-блоків в контексті модульного додавання. Зокрема, про вид та структуру DDT та класів DDT-еквівалентності.

Запропоновано рекурсивний алгоритм для пошуку S-блоку з заданою DDT на основі стратегії Backtracking. На основі цього алгоритму з урахуванням властивостей афінних перетворень в умовах модульного додавання представлено алгоритм отримання всього класу, так званих

афінних зсувів еквівалентності. Розроблено удосконалену щодо використання пам'яті версію цього алгоритму. Наведено асимптотичні оцінки складності алгоритму.

3 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

В третьому розділі описано деякі результати експериментальних досліджень, що отримані під час дослідження структури таблиць розподілу диференціалів у випадку диференціалів за модульним додаванням та не ввійшли до розділу 2 як аналітичні результати. Описані результати можуть бути корисними для подальшого дослідження цієї проблеми. До цих результатів відносяться: структура класів еквівалентності трьохбітових та чотирьохбітових S-блоків. Після чого наведено опис програмної реалізації запропонованих у розділі 2 оптимізованої та не оптимізованої версій алгоритму по відновленню класу афінних зсувів еквівалентності. Представлено модифікацію алгоритму відновлення, яка відновлює цілий клас DDT-еквівалентності.

3.1 Опис структури трьохбітових S-блоків

Майже всі аналітичні висновки, наведені у розділі 2, були отримані емпіричним шляхом та є узагальненнями часткових практичних випадків. В основному експериментальні дослідження було проведено на S-блоках малих бітностей через обмеження обчислювальних ресурсів для дослідження. Більшість результатів було отримано при дослідженні 3-ьох бітових S-блоків, що опинилися найкращим кандидатом для цих цілей, адже їх кількість не виходить за межі обчислювальних ресурсів звичайного персонального комп'ютера, та в той самий час вони мають достатньо статистики, щоб можна було простежити певні залежності для подальшого їх узагальнення.

Так, було встановлено, що не всі класи DDT-еквівалентності мають однакову потужність. Наприклад, класи DDT-еквівалентності 3-ьох бітових S-блоків діляться на 5 класів за кількістю S-блоків що входять до класу DDT-еквівалентності. Ці класи мають наступні потужності класів DDT-еквівалентності S-блоків: 8, 16, 32, 64, 128.

Для того, щоб не заплутувати читача далі в цій роботі, класи класів DDT-еквівалентності за потужностями класів DDT-еквівалентності будуть називатися *метакласами*.

Такий феномен щодо потужностей класів DDT-еквівалентності відбувається через те, що в межах цих класів S-блоки мають однакову чи схожу структуру, та в деяких класах DDT-еквівалентності в силу структури S-блоків не всі афінні зсуви та обернені зсуви не утворюють унікальні S-блоки.

Так, наприклад, завжди усі лінійні S-блоки сконцентровуються в метакласі з найменшою потужністю класів DDT-еквівалентності. У випадку 3-ьох бітових S-блоків усі їх класи DDT-еквівалентності утворюють свій метаклас, потужність якого 4, та всі класи DDT-еквівалентності в межах цього метакласу мають потужність рівну 8. Розглянемо довільний 3-ьох бітовий лінійний S-блок $S(x)$, що має наступний вигляд $S(x) = ax + b$, де коефіцієнт $a \in \mathbb{Z}_8^*$ та коефіцієнт $b \in \mathbb{Z}_8$. Так, виходячи з твердження 2.2, в результаті усіх можливих комбінацій афінних зсувів входів та виходів та усіх можливих таких комбінацій у випадку обернених афінних зсувів потужність отриманого класу становить 8 S-блоків. Та потужність цього метакласу пояснюється властивостями афінних зсувів та потужністю мультипликативної групи $\mathbb{Z}_8^*$.

Зазначимо, що для 3-ьох бітових S-блоків клас афінних зсувів еквівалентності повністю співпадає з класом DDT-еквівалентності.

Проте це не є справедливим для всіх бітностей S-блоків. Так, наприклад, для 4-ьох бітових S-блоків існують випадки, коли два S-блоки належать до одного класу DDT-еквівалентності, проте не є

афінно-еквівалентними. Прикладом можуть слугувати наступні 4-ьох бітові S-блоки:

$$S_1(x) = (0, 5, 3, 14, 6, 4, 12, 2, 8, 13, 10, 1, 9, 7, 15, 11),$$
$$S_2(x) = (8, 15, 4, 9, 7, 13, 5, 3, 11, 6, 12, 1, 14, 10, 2, 0).$$

S-блоки $S_1(x)$ та $S_2(x)$ є твірними елементами для 2 класів афінних зсувів еквівалентності, що не перетинаються, по 512 S-блоків кожний.

Так як наведений в розділі 2 алгоритм оптимізований припущенням $S(0) = 0$, в ході досліджень було підраховано кількість таких S-блоків в класі DDT-еквівалентності в залежності від потужності метакласу. Результати наведені в таблиці 3.1, де $|M|$ - потужність метакласу та $\#S(x)$ - кількість S-блоків:

Таблиця 3.1 – Залежність кількості S-блоків в класі DDT-еквівалентності, що $S(0) = 0$, в залежності від розмірності метакласу

$\|M\|$	8	16	32	64	128
$\#S(x)$	1	2	4	8	16

Серед усього іншого було помічено, що, чим більше потужність метакласу, до якого входить клас DDT-еквівалентності, тим менше нульових елементів в DDT, та, як наслідок, тим більше частота трапляння дрібних елементів серед значень комірок DDT.

3.2 Програмна реалізація алгоритму відновлення класу афінних зсувів еквівалентності

В додатку 3.2 можна ознайомитись з програмною реалізацією двох варіацій алгоритму відновлення класу афінних зсувів еквівалентності, наведеного у розділі 2, а саме оптимізована та не оптимізована з точки зору пам'яті варіації. Даний алгоритм відновлення реалізований на мові програмування Java в рекурсивній та в ітеративній варіції.

Представлений алгоритм відновлення був розроблений на персональному комп'ютері (далі ПК), що має наступні обчислювальні ресурси: CPU 2.3 GHz, RAM 8 GB. Усі подальші зауваження щодо роботи алгоритму та швидкості роботи алгоритму відносяться виключно до описаної конфігурації обчислювальних ресурсів.

Зауважимо, що не рекомендується реалізовувати описаний алгоритм відновлення рекурсивним чином, адже такий варіант не є оптимальною реалізацією з точки зору пам'яті, так як для S-блоків великих бітностей він призводить до переповнення стеку викликів функцій, в наслідок чого алгоритм завершує свою роботу передчасно та не дає жодних результатів. Так, на представленому ПК при відновлюванні S-блоків великих бітностей (більше 6) виникає проблема переповнення стеку викликів функцій. Оскільки будь-який рекурсивний алгоритм може бути реалізований ітеративно, цей алгоритм був реалізований не рекурсивно задля наведених вище оптимізацій.

Наведемо приклад роботи програми для 3-ьохбітового нелінійного S-

блока $S(x) = (1, 2, 7, 0, 5, 6, 3, 4)$, що має наступну DDT:

$$\begin{pmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \end{pmatrix}$$

Тоді на виході нашого алгоритму як результат отримаємо наступну множину S-блоків:

$$\{(5, 6, 3, 4, 1, 2, 7, 0), (6, 3, 4, 1, 2, 7, 0, 5), (3, 4, 1, 2, 7, 0, 5, 6),$$
$$(0, 5, 6, 3, 4, 1, 2, 7), (7, 0, 5, 6, 3, 4, 1, 2), (4, 1, 2, 7, 0, 5, 6, 3),$$
$$(3, 0, 1, 6, 7, 4, 5, 2), (1, 2, 7, 0, 5, 6, 3, 4), (2, 7, 0, 5, 6, 3, 4, 1),$$
$$(0, 1, 6, 7, 4, 5, 2, 3), (1, 6, 7, 4, 5, 2, 3, 0), (2, 3, 0, 1, 6, 7, 4, 5),$$
$$(7, 4, 5, 2, 3, 0, 1, 6), (4, 5, 2, 3, 0, 1, 6, 7), (5, 2, 3, 0, 1, 6, 7, 4),$$
$$(6, 7, 4, 5, 2, 3, 0, 1)\}.$$

Як бачимо, серед виходу нашого алгоритму присутній шуканий S-блок.

Також в рамках виконання практичної частини була виконана модифікація Backtracking алгоритму з розділу 2. На відміну від розглянутого алгоритму дана модифікація відновлює повний клас DDT-еквівалентності, а не лише клас афінних зсувів еквівалентності. В даній модифікації не використовується початкове припущення $S[0] = 0$.

Через це починається перебір з нульового рівня, та фіксація елементу на кожному рівні відбувається з урахуванням значення S-блоку в нулі, тобто для a-того елементу ($a \geq 1$) вираз матиме вигляд: $S(a) = b + S(0)$. Очевидно, що складність такої модифікації є більшою, адже виконуються додаткові обчислення.

Так як асимптотичні оцінки для запропонованого алгоритму не є змістовними, наведемо фактичні вимірювання на описаному вище персональному комп'ютері у вигляді залежності середньої швидкості відновлювання від бітності S-блоків у таблиці 3.2, де алгоритм з розділу 2 називається Backtracking, його оптимізація з точки зору пам'яті, що описана в тому ж розділі, – Backtracking (оптимізований), реалізація відновлення класу афінних зсувів еквівалентності, що описана в розділі 3, – Backtracking (повний):

Таблиця 3.2 – Залежність часу роботи алгоритмів від бітності відновлювального S-блока

Реалізація Бітність S-блоку	2	3	4	5
Повний перебір	0.1667	135.86	173659156070*	$1,7892897e+34$*
Backtracking	0.1667	0.15	105	128.0
Backtracking (оптимізований)	0.04167	0.5	23	37.0
Backtracking (повний)	0.083	1.57	18256	$1.0183163e+31$*

Всі виміри, що відмічені позначкою *, мають неточні значення. Ці вимірювання були отримані наступним чином: було заміряно швидкість роботи алгоритму через рівну кількість перебраних S-блоків та усереднені, після чого на їх основі робились приблизні розрахунки загального часу роботи алгоритму. Для повного перебору оцінка, отримана таким чином, має меншу похибку, адже його робота рівномірна,

в той час як для Backtracking даний підхід має більшу похибку через нерівномірний час, що витрачається на перевірку різних варіантів.

Так, можемо побачити, що повний перебір не є застосовним для цієї задачі для S-блоків з бітностю більше за 3. Backtracking (повний), що відновлює цілий клас DDT-еквівалентності, показує значно кращі результати, проте опиняється теж незастосовним для бітностей більше 4. Алгоритми Backtracking та Backtracking (оптимізований) мають набагато кращі показники, проте слід пам'ятати, що вони не відновлюють цілий клас DDT-еквівалентності, а лише шукають твірний елемент та утворюють клас афінних зсувів еквівалентності.

Висновки до розділу 3

Отже, в ході виконання експериментальних досліджень, окрім отриманих аналітичних результатів наведених у розділі 2, отримано часткові результати для 3-ьох та 4-ьох бітових S-блоків. Зокрема, встановлено, що класи DDT-еквівалентності для 3-ьох бітових S-блоків можна згрупувати в, так звані, метакласи, яких для 3-ьох бітових S-блоків налічується 5, досліджено структуру класів DDT-еквівалентності для 4-ьох бітових S-блоків, а саме встановлено, що клас афінних зсувів еквівалентності є підкласом класу DDT-еквівалентності.

Розроблено програмну реалізацію оптимізованої та неоптимізованої варіації алгоритму відновлення класу афінних зсувів еквівалентності. Розроблено програмну реалізацію модифікації, що відновлює повний клас DDT-еквівалентності теж на основі стратегії Backtracking. Проведено порівняння швидкості роботи всіх версій Backtracking алгоритму та наведено оцінки часу роботи для повного перебора.

ВИСНОВКИ

В ході виконання дослідження проведено огляд та аналіз опублікованих джерел за темою проблеми відновлення S-блока за таблицею розподілу диференціалів, який показав, що дана проблема є доволі добре розв'язана для випадку диференціалів, побудованих за побітовим додаванням, зокрема, був розроблений алгоритм, що ефективно вирішує дане питання. Однак в оглянутих джерелах не знайдено жодної інформації щодо розв'язання даної проблеми у випадку диференціалів побудованих за модулем 2^n.

З огляду на це проведено власне дослідження задачі відновлення S-блоку за таблицею розподілу диференціалів за модульним додаванням. В ході цього дослідження описано клас лінійних S-блоків у контексті диференціалів, що обчислюються за модульним додаванням. Як результат, описано загальний вид та основні властивості DDT лінійних S-блоків та класів DDT-еквівалентності, деякі з властивостей лінійних S-блоків узагальнено на випадок DDT довільних S-блоків та, таким чином, сформовано основні властивості DDT відносно модульного додавання.

Досліджено афінні перетворення над S-блоками у випадку модульного додавання з точки зору впливу на вид DDT та, як результат, встановлено, що певні класи афінних перетворень над $\mathbb{Z}_{2^n}$ не змінюють вид DDT S-блоків, зокрема, досліджено, що до таких перетворень відносяться класи, так званих, афінних зсувів та класи обернених афінних зсувів аргументів та виходів S-блоків, які утворюють спеціальні класи афінних зсувів еквівалентності. Помічено, що для S-блоків малої бітності класи DDT-еквівалентності повністю співпадають з класами афінних зсувів еквівалентності, проте для S-блоків великих бітностей помічено, що існують класи DDT-еквівалентності, що складаються з класів афінних зсувів еквівалентності, що не перетинаються.

Досліджено основні властивості структури класів DDT-еквівалентності. Так, помічено, що класи DDT-еквівалентності мають різні потужності. Наведені властивості значно ускладнюють задачу відновлення класу DDT-еквівалентності за представником. Також в роботі зібрані різного роду статистичні відомості про таблиці розподілу диференціалів S-блоків малих бітностей.

Як результат описаних досліджень представлено алгоритм відновлення класу афінних зсувів еквівалентності за DDT на основі стратегії Backtracking, та проведено аналіз його часової складності. Розроблена програмна реалізація цього алгоритму на мові програмування Java. Представлені модифікації описаного алгоритму, зокрема, модифікація, що використовує менше пам'яті та має кращі показники швидкості роботи. Та інша модифікація, що відновлює цілий клас DDT-еквівалентності за час набагато менший, ніж повний перебір.

Не дивлячись на отримані результати та той факт, що поставлена задача була розв'язана в ході виконання роботи, існує ряд напрямків дослідження, що не були цілковито висвітлені. Так, структура класів DDT-еквівалентності для S-блоків 3-ьох та більше бітностей є досі не вивченою до кінця. Серед іншого слід звернути уваги на афінні перетворення для S-блоків великих бітностей та знайти зв'язок між класами афінних зсувів еквівалентності в межах класу DDT-еквівалентності для S-блоків з бітністю більше за 4. Слід також дослідити питання потужності класів афінних зсувів для різних S-блоків та навчитися розрізняти такі S-блоки за метакласами. Довести чи спростувати припущення, що в межах метакласу усі S-блоки мають однакову структуру.

Детальне вивчення цього напрямку може дозволити розробити більш швидкі модифікації описаного алгоритму чи навіть розв'язати проблему відновлення S-блоку за DDT не на основі стратегії Backtracking, що може бути значно оптимальнішим розв'язком.

ПЕРЕЛІК ПОСИЛАНЬ

1. Завадська Л.О., Савчук М.М. Математичні методи захисту інформації: курс лекцій. Частина 1. – Київ: НТУУ «КПІ», 2008. – Ч.1. – 128 с.

2. Логачев О.А., Сальников А.А., Ященко В.В. Булевы функции в теории кодирования и криптологии – М.: МЦНМО, 2004. – 470с.

3. Фомичев В.М. Дискретная математика и криптология. – М.: ДИАЛОГ-МИФИ, 2003 – 400с.

4. Knudsen, Lars R., Robshaw, Matthew. The Block Cipher Companion. – Springer, 2011. – 279 pp. – ISBN 978-3-642-17342-4

5. Dunkelman Orr, Huang Senyang. Reconstructing an S-box from its Difference Distribution Table. — Cryptology ePrint Archive, Report 2018/811. — 2018. — https://eprint.iacr.org/2018/811.

6. Two notions of differential equivalence on Sboxes / Christina Boura, Anne Canteaut, Jérémy Jean, Valentin Suder // Designs, Codes and Cryptography. — 2019. — Mar. — Vol. 87, no. 2. — P. 185– 202. — Access mode: https://doi.org/10.1007/ s10623-018-0496-z.

7. Chabaud Florent, Vaudenay Serge. Links between differential and linear cryptanalysis // Advances in Cryptology — EUROCRYPT'94 / Ed. by Alfredo De Santis. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1995. — P. 356–365.

8. Blondeau Céline, Nyberg Kaisa. New Links Between Differential and Linear Cryptanalysis. — Cryptology ePrint Archive, Report 2015/183. — 2015. — https: //eprint.iacr.org/2015/183.

9. Blondeau Céline, Leander Gregor, Nyberg Kaisa. Differential-Linear Cryptanalysis Revisited // Journal of Cryptology. — 2017. — Jul. — Vol. 30, no. 3. — P. 859–888. — Access mode: https://doi.org/10. 1007/s00145-016-9237-5.

ДОДАТОК А ПСЕВДОКОД АЛГОРИТМУ ВГАДУВАННЯ ЕЛЕМЕНТУ S-БЛОКУ

Algorithm 0.1 Вгадування елементу S-блоку на a-тій позиції (частина 1)

Вхід: Позиція S-блока a, що вгадується; Родина множин ненульових елементів рядків DDT $\{\mathbb{A}_a : 0 \leq a \leq n\}$; Родина множин перевіренних варіантів $\{\mathbb{T}_a : 0 \leq a \leq n\}$; Таблиця розподілу диференціалів ddt;

Вихід: Один із твірних елементів класу афінних зсувів еквівалентності, з DDT ddt

якщо $\mathbb{A}_a = \mathbb{T}_a$ **тоді**
    $\mathbb{T}_a = \varnothing$;
    прирівняти елемент S-блоку $S(a)$ до 0;
    рекурсивно викликати процедуру з параметрами $\{a - 1;\ \{\mathbb{A}_a : 0 \leq a \leq n\};\ \{\mathbb{T}_a : 0 \leq a \leq n\};$ ddt$\}$
кінець умови

цикл *для всіх* $x' \in \mathbb{A}_a$ **виконати**
    покласти x' до $\mathbb{T}_a$
    якщо x' *немає серед вгаданих елементів S-блоку* $S(x)$ **тоді**
        прирівняти елемент S-блоку $S(a)$ до x';
        вийти з циклу досроково;
    кінець умови
кінець циклу

якщо $S[a] = 0$ **тоді**
    $\mathbb{T}_a = \varnothing$;
    рекурсивно викликати процедуру з параметрами $\{a - 1;\ \{\mathbb{A}_a : 0 \leq a \leq n\};\ \{\mathbb{T}_a : 0 \leq a \leq n\};$ ddt$\}$;
кінець умови

Algorithm 0.2 Вгадування елементу S-блоку на a-тій позиції (частина 2)

якщо $a = n - 1$ **тоді**

  якщо *S-блок* $S(x)$ *має DDT ddt* **тоді**

    повернути $S(x)$

  інакше

    $\mathbb{T}_a = \varnothing$

    прирівняти елемент S-блоку $S(a)$ до 0

    рекурсивно викликати процедуру з параметрами $\{a - 1;\ \{\mathbb{A}_a : 0 \leq a \leq n\};$ $\{\mathbb{T}_a : 0 \leq a \leq n\};$ ddt$\};$

  кінець умови

інакше

  цикл $0 < x < a$ **виконати**

    якщо $\delta(a, S(a) - S(k)) = 0$ **тоді**

      рекурсивно викликати процедуру з параметрами $\{a;\ \{\mathbb{A}_a : 0 \leq a \leq n\};$ $\{\mathbb{T}_a : 0 \leq a \leq n\};$ ddt$\};$

    кінець умови

  кінець циклу

кінець умови

рекурсивно викликати процедуру з параметрами $\{a + 1;\ \{\mathbb{A}_a : 0 \leq a \leq n\};$ $\{\mathbb{T}_a : 0 \leq a \leq n\};$ ddt$\};$

ДОДАТОК Б КОДИ ПРОГРАМ

```java
package ua.kpi.ipt.service;

import ua.kpi.ipt.model.DDT;
import ua.kpi.ipt.model.SBox;

import java.util.ArrayList;
import java.util.List;

import static ua.kpi.ipt.Constants.mod;

public class SBoxService {

    public DDT calculateDDT(SBox sBox) {
        int[][] ddt = new int[mod][mod];

        for (int i = 0; i < mod; i++) {
            for (int j = 0; j < mod; j++) {
                for (int k = 0; k < mod; k++) {
                    if (sBox.substitute((k + i) % mod)
                            == (j + sBox.substitute(k)) % mod) {
                        ddt[i][j]++;
                    }
                }
            }
        }

        return new DDT(ddt);
    }

    public List<SBox> generateDDTEquivalenceClass(SBox generator) {
        List<SBox> result = new ArrayList<>();

        for (int i = 0; i < mod; i++) {
            for (int j = 0; j < mod; j++) {
                result.add(outputAffineTransformation(argumentAffineTransformation(generator, mod -
                ↪  1, j), mod - 1, i));
```

```
36                  result.add(outputAffineShift(argumentAffineShift(generator, i), j));
37              }
38          }
39
40          return result;
41      }
42
43
44      private SBox argumentAffineTransformation(SBox s, int a, int b) {
45          int[] table = s.getSubstituteTable();
46          int[] newTable = new int[mod];
47
48          for (int i = 0; i < mod; i++) {
49              newTable[i] = table[(a * i + b) % mod];
50          }
51
52          return new SBox(newTable);
53      }
54
55      private SBox outputAffineTransformation(SBox s, int a, int b) {
56          int[] table = s.getSubstituteTable();
57          int[] newTable = new int[mod];
58
59          for (int i = 0; i < mod; i++) {
60              newTable[i] = (a * table[i] + b) % mod;
61          }
62
63          return new SBox(newTable);
64      }
65
66      private SBox argumentAffineShift(SBox s1, int a) {
67          int[] table = s1.getSubstituteTable();
68          int[] newTable = new int[mod];
69
70          for (int i = 0; i < mod; i++) {
71              newTable[i] = table[(i + a) % mod];
72          }
73
74          return new SBox(newTable);
```

```
75      }
76
77      private SBox outputAffineShift(SBox s, int a) {
78          int[] table = s.getSubstituteTable();
79          int[] newTable = new int[mod];
80
81          for (int i = 0; i < mod; i++) {
82              newTable[i] = (table[i] + a) % mod;
83          }
84
85          return new SBox(newTable);
86      }
87
88  }
```

```
1   package ua.kpi.ipt.model;
2
3   import java.util.Arrays;
4
5   public class SBox {
6
7       private int[] substituteTable;
8
9       public SBox(int[] table) {
10          this.substituteTable = new int[table.length];
11          System.arraycopy(table, 0, this.substituteTable, 0, table.length);
12      }
13
14      public int substitute(int input) {
15          if (input >= substituteTable.length || input < 0) {
16              throw new IllegalArgumentException();
17          }
18          return substituteTable[input];
19      }
20
21      public int[] getSubstituteTable() {
22          return substituteTable;
23      }
```

```
24
25      @Override
26      public boolean equals(Object o) {
27          if (this == o) return true;
28          if (o == null || getClass() != o.getClass()) return false;
29          SBox sBox = (SBox) o;
30          return Arrays.equals(substituteTable, sBox.substituteTable);
31      }
32
33      @Override
34      public int hashCode() {
35          return Arrays.hashCode(substituteTable);
36      }
37
38      @Override
39      public String toString() {
40          return Arrays.toString(substituteTable);
41      }
42  }
```

```
1   package ua.kpi.ipt.model;
2
3   import ua.kpi.ipt.utils.ArrayUtils;
4
5   import java.util.Arrays;
6
7   public class DDT {
8
9       private int[][] ddt;
10
11      public DDT(int[][] ddt) {
12          this.ddt = new int[ddt.length][ddt.length];
13          for (int i = 0; i < ddt.length; i++) {
14              System.arraycopy(ddt[i], 0, this.ddt[i], 0, ddt.length);
15          }
16      }
17
18      public int[][] getDdt() {
```

```
19          return ddt;
20      }
21
22      @Override
23      public boolean equals(Object o) {
24          if (this == o) return true;
25          if (o == null || getClass() != o.getClass()) return false;
26          DDT ddt1 = (DDT) o;
27          boolean result = true;
28          for (int i = 0; i < ddt.length; i++) {
29              if (!Arrays.equals(ddt[i], ddt1.ddt[i])) {
30                  result = false;
31                  break;
32              }
33          }
34          return result;
35      }
36
37      @Override
38      public int hashCode() {
39          int[] res = ddt[0];
40          for (int i = 1; i < ddt.length; i++) {
41              res = ArrayUtils.concatArrays(res, ddt[i]);
42          }
43          return Arrays.hashCode(res);
44      }
45
46      @Override
47      public String toString() {
48          StringBuilder sb = new StringBuilder();
49          for (int i = 0; i < ddt.length; i++) {
50              for (int j = 0; j < ddt.length; j++) {
51                  sb.append(ddt[i][j]).append(" ");
52              }
53              sb.append('\n');
54          }
55          return sb.toString();
56      }
57  }
```

```java
package ua.kpi.ipt.utils;

import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class ArrayUtils {

    private static Random random = new Random();

    public static int[] concatArrays(int[] arr1, int[] arr2) {
        return IntStream.concat(Arrays.stream(arr1), Arrays.stream(arr2)).toArray();
    }

    public static boolean elementIsNotPresentInArray(int[] table, int el) {
        for (int i : table) {
            if (i == el)
                return false;
        }
        return true;
    }

    public static int[][] getNonZeroIndexesTable(int[][] table) {
        int[][] res = new int[table.length][];
        int[] temp = new int[table.length];
        for (int i = 0; i < table.length; i++) {
            int count = 0;
            for (int j = 0; j < table.length; j++) {
                if (table[i][j] != 0) {
                    temp[count++] = j;
                }
            }
            res[i] = new int[count];
            System.arraycopy(temp, 0, res[i], 0, count);
            Arrays.fill(temp, 0);
```

```
        }
        return res;
    }

    public static int[] tripleShuffle(int[] arr) {
        return shuffle(shuffle(shuffle(arr)));
    }


    public static int[] shuffle(int[] arr) {
        List<Integer> list = IntStream.range(0, arr.length).boxed().collect(Collectors.toList());
        for (int i = 0; i < arr.length; i++) {
            swap(arr, i, getAndRemove(list));
        }
        return arr;
    }

    private static int getAndRemove(List<Integer> list) {
        Integer integer = list.get(random.nextInt(list.size()));
        list.remove(integer);
        return integer;
    }


    public static void swap(int[] arr, int index1, int index2) {
        int buff = arr[index1];
        arr[index1] = arr[index2];
        arr[index2] = buff;
    }
}
```

```
package ua.kpi.ipt;

public interface Constants {

    int mod = 64;

}
```

```java
package ua.kpi.ipt.recovery;

import ua.kpi.ipt.model.DDT;
import ua.kpi.ipt.model.SBox;

import java.util.List;

public interface RecoveryAlgorithm {

    List<SBox> recover(DDT ddt);

}
```

```java
package ua.kpi.ipt.recovery;

import ua.kpi.ipt.Constants;
import ua.kpi.ipt.model.DDT;
import ua.kpi.ipt.model.SBox;
import ua.kpi.ipt.service.SBoxService;
import ua.kpi.ipt.utils.ArrayUtils;

import java.util.*;

public class Backtracking implements RecoveryAlgorithm {

    private SBoxService sBoxService;

    public Backtracking(SBoxService sBoxService) {
        this.sBoxService = sBoxService;
    }

    @Override
    public List<SBox> recover(DDT ddt) {
        int[][] table = ddt.getDdt();

        int[] stable = new int[Constants.mod];
        stable[0] = 0;

```

```
26          List<List<Integer>> nonZeros = getNonZeroIndexes(table);
27          List<Set<Integer>> tries = new ArrayList<>();
28          for (int i = 0; i < nonZeros.size(); i++) {
29              tries.add(new HashSet<>());
30          }
31
32          for (int a = 0; a < Constants.mod - 1; a++) {
33
34              List<Integer> shots = nonZeros.get(a);
35              Set<Integer> wastedShots = tries.get(a);
36
37              if (wastedShots.size() == shots.size()) {
38                  tries.get(a).clear();
39                  stable[a] = 0;
40                  a -= 2; continue;
41              }
42
43              for (int num : shots) {
44                  if (!wastedShots.contains(num)) {
45                      wastedShots.add(num);
46                      if (ArrayUtils.elementIsNotPresentInArray(stable, num)) {
47                          stable[a + 1] = num; break;
48                      }
49                  }
50              }
51              if (stable[a + 1] == 0) {
52                  tries.get(a).clear();
53                  stable[a] = 0;
54                  a -= 2; continue;
55              }
56
57              if (a == Constants.mod - 2) {
58                  SBox sBox = new SBox(stable);
59                  DDT candidate = sBoxService.calculateDDT(sBox);
60                  if (!candidate.equals(ddt)) {
61                      tries.get(a).clear();
62                      stable[a+1] = 0;
63                      stable[a] = 0;
64                      a -= 2;
```

```
65                  }
66              } else {
67                  for (int b = 1; b <= a; b++) {
68                      if (table[a - b + 1][(stable[a + 1] - stable[b] + Constants.mod) %
                        ↪  Constants.mod] == 0) {
69                          stable[a + 1] = 0;
70                          a--; break;
71                      }
72                  }
73              }
74          }
75
76          return sBoxService.generateDDTEquivalenceClass(new SBox(stable));
77      }
78
79      private List<List<Integer>> getNonZeroIndexes(int[][] table) {
80          List<List<Integer>> nonZeros = new ArrayList<>();
81
82          for (int i = 1; i < Constants.mod; i++) {
83              List<Integer> line = new ArrayList<>();
84              for (int j = 0; j < Constants.mod; j++) {
85                  if (table[i][j] != 0) {
86                      line.add(j);
87                  }
88              }
89              nonZeros.add(line);
90          }
91
92          return nonZeros;
93      }
94  }
```

```
1   package ua.kpi.ipt.recovery;
2
3   import ua.kpi.ipt.model.DDT;
4   import ua.kpi.ipt.model.SBox;
5   import ua.kpi.ipt.service.SBoxService;
6   import ua.kpi.ipt.utils.ArrayUtils;
```

```
7
8  import java.util.ArrayList;
9  import java.util.Arrays;
10 import java.util.List;
11
12 import static ua.kpi.ipt.Constants.mod;
13
14 public class BacktrackingFull implements RecoveryAlgorithm {
15
16     public static final int FILLER = -1;
17     public static final int LAST_POSITION = mod - 1;
18     private SBoxService sBoxService;
19
20     public BacktrackingFull(SBoxService sBoxService) {
21         this.sBoxService = sBoxService;
22     }
23
24     @Override
25     public List<SBox> recover(DDT ddt) {
26         int[][] table = ddt.getDdt();
27
28         int[] stable = new int[mod];
29
30         for (int i = 0; i < mod; i++) {
31             stable[i] = FILLER;
32         }
33
34         int[][] nonZeroIndexes = ArrayUtils.getNonZeroIndexesTable(table);
35
36         int offset = 0;
37
38         List<SBox> result = new ArrayList<>();
39
40         for (int j = 0; j < mod; j++) {
41             layer:
42             for (int a = 0; a < mod; a++) {
43                 el:
44                 for (int i = offset; i < nonZeroIndexes[a].length; i++) {
45                     offset = 0;
```

```
46                      int el = (nonZeroIndexes[a][i] + j) % mod;
47                      if (ArrayUtils.elementIsNotPresentInArray(stable, el)) {
48                          stable[a] = el;
49                          if (a == LAST_POSITION) {
50                              SBox sBox = new SBox(stable);
51                              if (sBoxService.calculateDDT(sBox).equals(ddt)) {
52                                  result.add(sBox);
53                              }
54                              stable[a] = FILLER;
55                              offset = getOffset(nonZeroIndexes[a - 1], (stable[a - 1] - j + mod) %
                                ↪  mod);
56                              a -= 2;
57                              continue layer;
58                          } else {
59                              for (int b = 0; b < a; b++) {
60                                  if (table[a - b][(stable[a] - stable[b] + mod) % mod] == 0) {
61                                      //System.out.println(j);
62                                      continue el;
63                                  }
64                              }
65                          }
66                          continue layer;
67                      }
68                  }
69                  if (a == 0) {
70                      break;
71                  }
72                  stable[a] = FILLER;
73                  offset = getOffset(nonZeroIndexes[a - 1], (stable[a - 1] - j + mod) % mod);
74                  a -= 2;
75              }
76          }
77          return result;
78      }
79
80      private int getOffset(int[] nonZeroIndex, int key) {
81          return Arrays.binarySearch(nonZeroIndex, key);
82      }
83  }
```

```java
package ua.kpi.ipt.recovery;

import ua.kpi.ipt.Constants;
import ua.kpi.ipt.model.DDT;
import ua.kpi.ipt.model.SBox;
import ua.kpi.ipt.service.SBoxService;
import ua.kpi.ipt.utils.ArrayUtils;

import java.util.*;

public class BacktrackingOptimized implements RecoveryAlgorithm {

    private SBoxService sBoxService;

    public BacktrackingOptimized(SBoxService sBoxService) {
        this.sBoxService = sBoxService;
    }

    @Override
    public List<SBox> recover(DDT ddt) {
        int[][] table = ddt.getDdt();

        int[] stable = new int[Constants.mod];

        int[][] nonZeroIndexes = ArrayUtils.getNonZeroIndexesTable(table);

        int offset = 0;

        layer:
        for (int a = 1; a < Constants.mod; a++) {

            //System.out.println(Arrays.toString(stable));
            el:
            for (int i = offset; i < nonZeroIndexes[a].length; i++) {
                offset = 0;
                if (ArrayUtils.elementIsNotPresentInArray(stable, nonZeroIndexes[a][i])) {
                    stable[a] = nonZeroIndexes[a][i];
```

```
38                      if (a == Constants.mod - 1) {
39                          SBox sBox = new SBox(stable);
40                          DDT candidate = sBoxService.calculateDDT(sBox);
41                          if (!candidate.equals(ddt)) {
42                              stable[a] = 0;
43                              offset = Arrays.binarySearch(nonZeroIndexes[a - 1], stable[a - 1]);
44                              a -= 2;
45                              continue layer;
46                          }
47                      } else {
48                          for (int b = 1; b < a; b++) {
49                              if (table[a - b][(stable[a] - stable[b] + Constants.mod) %
                                ↪  Constants.mod] == 0) {
50                                  continue el;
51                              }
52                          }
53                      }
54                      continue layer;
55                  }
56              }
57              stable[a] = 0;
58              offset = Arrays.binarySearch(nonZeroIndexes[a - 1], stable[a - 1]);
59              a -= 2;
60          }
61
62          return sBoxService.generateDDTEquivalenceClass(new SBox(stable));
63      }
```

```
1   package ua.kpi.ipt.recovery;
2
3
4   import ua.kpi.ipt.model.DDT;
5   import ua.kpi.ipt.model.SBox;
6   import ua.kpi.ipt.service.SBoxService;
7   import ua.kpi.ipt.utils.ArrayUtils;
8
9   import java.util.ArrayList;
10  import java.util.List;
```

```java
import java.util.stream.IntStream;

import static ua.kpi.ipt.Constants.mod;

public class BrutForce implements RecoveryAlgorithm {

    private SBoxService sBoxService;


    public BrutForce(SBoxService sBoxService) {
        this.sBoxService = sBoxService;
    }

    @Override
    public List<SBox> recover(DDT ddt) {
        List<SBox> result = new ArrayList<>();

        permuteInList(IntStream.range(0, mod).toArray(), result, 0, mod-1, ddt);

        return result;
    }

    private void permuteInList(int[] arr, List<SBox> list, int l, int r, DDT ddt) {
        if (l == r) {
            SBox sBox = new SBox(arr);
            if (sBoxService.calculateDDT(sBox).equals(ddt)) {
                list.add(sBox);
            }
        } else {
            for (int i = l; i <= r; i++) {
                ArrayUtils.swap(arr, l, i);
                permuteInList(arr, list, l + 1, r, ddt);
                ArrayUtils.swap(arr, l, i);
            }
        }
    }
```